

UT/EL	BSC/NT
project:	
date:	KT. 1989

# **BASIC/UX Interfacing Techniques**

## **Vol. 2: Specific Interfaces**

**HP 9000 Series 300 Computers  
BASIC/UX 5.5**

HP Part Number 98796-90030



**HEWLETT  
PACKARD**

**Hewlett-Packard Company**

3404 East Harmony Road, Fort Collins, Colorado 80525

## NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MANUAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

## WARRANTY

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

Copyright © Hewlett-Packard Company 1988

This document contains information which is protected by copyright. All rights are reserved. Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

### Restricted Rights Legend

Use, duplication or disclosure by the U.S. Government Department of Defense is subject to restrictions as set forth in paragraph (b)(3)(ii) of the Rights in Technical Data and Software clause in FAR 52.227-7013.

Use of this manual and flexible disc(s) or tape cartridge(s) supplied for this pack is restricted to this product only. Additional copies of the programs can be made for security and back-up purposes only. Resale of the programs in their present form or with alterations, is expressly prohibited.

Copyright © AT&T, Inc. 1980, 1984

Copyright © The Regents of the University of California 1979, 1980, 1983

This software and documentation is based in part on the Fourth Berkeley Software Distribution under license from the Regents of the University of California.



# Win an HP Calculator!

Your comments and suggestions help us determine how well we meet your needs. **Returning this card with your name and address enters you into a quarterly drawing for an HP calculator\*.**

## BASIC/UX Interfacing Techniques

	Agree				Disagree
The product was easy to install.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The manual is well organized.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
It is easy to find information in the manual.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The manual explains features well.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Step by step procedures are easy to perform.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Overall, the manual meets my expectations.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

**Have you installed HP-UX before?**

☐ Yes

☐ No

**Did installation require outside assistance?**

☐ Yes

☐ No

*fold* —

Please write additional comments, particularly if you disagree with a statement above. Use additional pages if you wish. The more specific your comments, the more useful they are to us.

**Comments:** \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\* Offer expires June 1990. (98796-90030 E0988.)

*Please Tape Here*

Please print or type your name and address.

Name: \_\_\_\_\_

Company: \_\_\_\_\_

Address: \_\_\_\_\_

City, State, Zip: \_\_\_\_\_

Telephone: \_\_\_\_\_

Additional Comments: \_\_\_\_\_

BASIC/UX Interfacing Techniques  
HP Part Number 98796-90030  
E0988



NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES

# BUSINESS REPLY MAIL

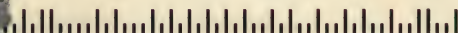
FIRST CLASS

PERMIT NO. 37

LOVELAND, COLORADO

POSTAGE WILL BE PAID BY ADDRESSEE

Hewlett-Packard Company  
Attn: Learning Products Center  
3404 East Harmony Road  
Fort Collins, Colorado 80525-9988



# Printing History

---

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages to be merged into the manual by the user. Each updated page will be indicated by a revision date at the bottom of the page. A vertical bar in the margin indicates the changes on each page. Note that pages which are rearranged due to changes on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

September 1988...Edition 1





# Table of Contents

---

## Chapter 10: Display Interfaces

Description of Displays .....	10-1
Display Specifics .....	10-2
How the Default Display Device Is Chosen .....	10-2
Overview of Display Features .....	10-3
Clearing the Screen .....	10-5
The Output Area and the Display Line .....	10-5
Determining Screenwidth and Other Attributes .....	10-7
Changing Pen Colors in Display Regions .....	10-8
Making Graphics Dominant .....	10-14
Output to the CRT .....	10-15
Numeric Outputs .....	10-15
String Outputs .....	10-16
Control Characters .....	10-16
Display-Enhancement Characters .....	10-19
The Display Functions Mode .....	10-21
Output-Area Memory .....	10-23
Determining Above-Screen Lines .....	10-23
The PRINT Position .....	10-25
Scrolling the Display .....	10-26
Entering from the CRT .....	10-28
Reading a Screen Line .....	10-28
Reading the Entire Output-Area Memory .....	10-29
Additional CRT Features .....	10-31
The DISP Line .....	10-31
Disabling the Cursor Character .....	10-32
Enabling the Insert Mode .....	10-33
Softkey Labels .....	10-34
Softkey Label Colors .....	10-36
Summary of CRT STATUS and CONTROL Registers .....	10-37

## **Chapter 11: The Keyboard Interface**

Overview .....	11-1
Keyboard Supported .....	11-1
HP 98203 Keyboard Compatibility Mode .....	11-2
Re-Configuring HIL Devices .....	11-2
Overview of Keyboard Features .....	11-2
ASCII and Non-ASCII Keys .....	11-3
The Shift and Control Keys .....	11-3
Keyboard Operating Modes .....	11-7
The Caps Lock Mode .....	11-7
The Print All Mode .....	11-8
Disabling Scrolling .....	11-8
Modifying the Repeat and Delay Intervals .....	11-9
Entering Data from the Keyboard .....	11-11
Sending the EOI Signal .....	11-13
Sending Data to the Keyboard .....	11-14
Sending Non-ASCII Keystrokes to the Keyboard .....	11-14
Second Byte of Non-ASCII Key Sequences (String) .....	11-16
Closure Keys .....	11-22
Softkeys .....	11-24
Sensing Knob Rotation .....	11-25
Enhanced Keyboard Control .....	11-27
Trapping Keystrokes .....	11-27
Mouse Keys .....	11-30
Softkeys and Knob Rotation .....	11-30
Disabling Interactive Keyboard .....	11-31
Locking Out the Keyboard .....	11-32
Special Considerations .....	11-33
Keyboard Status and Control Registers .....	11-34

## **Chapter 12: The HP-IB Interface**

Introduction .....	12-1
Initial Installation and Verification .....	12-2
Communicating with Devices .....	12-3
HP-IB Device Selectors .....	12-3
Moving Data Through the HP-IB .....	12-4
Using an Interface in the HP-UX Environment .....	12-6
General Structure of the HP-IB .....	12-8
Addressing Multiple Listeners .....	12-11
Secondary Addressing .....	12-11



General Bus Management .....	12-12
Remote Control of Devices .....	12-13
Locking Out Local Control .....	12-14
Enabling Local Control .....	12-14
Triggering HP-IB Devices .....	12-15
Clearing HP-IB Devices .....	12-15
Aborting Bus Activity .....	12-16
HP-IB Service Requests .....	12-16
Polling HP-IB Devices .....	12-19
Advanced Bus Management .....	12-21
The Message Concept .....	12-21
Types of Bus Messages .....	12-22
Explicit Bus Messages .....	12-27
HP-IB Message Mnemonics .....	12-30
The Computer As a Non-Active Controller .....	12-32
Determining Controller Status and Address .....	12-32
Changing the Controller's Address .....	12-34
Passing Control .....	12-34
Interrupts While Non-Active Controller .....	12-35
Addressing a Non-Active Controller .....	12-39
Requesting Service .....	12-41
Responding to Parallel Polls .....	12-42
Responding to Serial Polls .....	12-44
Interface-State Information .....	12-45
Servicing Interrupts that Require Data Transfers .....	12-46
HP-IB Control Lines .....	12-49
Handshake Lines .....	12-50
The Attention Line (ATN) .....	12-50
The Interface Clear Line (IFC) .....	12-51
The Remote Enable Line (REN) .....	12-51
HP-IB Control Lines (continued)	
The End or Identify Line (EOI) .....	12-51
The Service Request Line (SRQ) .....	12-52
Determining Bus-Line States .....	12-53
Summary of HP-IB STATUS and CONTROL Registers .....	12-54
HP-IB Status and Control Registers (cont.) .....	12-55
HP-IB Status and Control Registers (cont.) .....	12-56
HP-IB Status and Control Registers (cont.) .....	12-57
HP-IB Status and Control Registers (cont.) .....	12-58
HP-IB Status and Control Registers (cont.) .....	12-59

Summary of HP-IB READIO and WRITEIO Registers .....	12-60
READIO Registers .....	12-60
HP-IB WRITEIO Registers .....	12-67
Summary of Bus Sequences .....	12-73
ABORT .....	12-73
CLEAR .....	12-74
LOCAL .....	12-74
LOCAL LOCKOUT .....	12-74
PASS CONTROL .....	12-75
PPOLL .....	12-75
PPOLL CONFIGURE .....	12-75
PPOLL UNCONFIGURE .....	12-76
REMOTE .....	12-76
SPOLL .....	12-77
TRIGGER .....	12-77

### **Chapter 13: RS-232C Serial Interfaces**

Asynchronous Data Communication .....	13-2
Data Transfers Between Computer and Peripheral .....	13-5
Overview of Serial Interface Programming .....	13-6
Determining Operating Parameters .....	13-6
Serial Configuration for BASIC/UX .....	13-7
Using Program Control to Override Defaults .....	13-9
Transferring Data .....	13-12
Entering and Outputting Data .....	13-12
Modem Line Handshaking .....	13-13
Incoming Data Error Detection and Handling .....	13-14
Advanced Programming Information .....	13-16
Sending BREAK Messages .....	13-16
Using the Modem Control Register .....	13-16
Cable Options and Signal Functions .....	13-18
The DTE Cable .....	13-19
The DCE Cable .....	13-20
RS-232C / CCITT V24 .....	13-23
HP 98626 and HP 98644 Serial Interface STATUS and CONTROL Registers .....	13-25
HP 98644 Interface Differences .....	13-31
Hardware Differences .....	13-31
BASIC Differences .....	13-34
Series 300 Built-In 98644 Interface Differences .....	13-35

## **Chapter 14: Datacomm Interfaces**

Prerequisites .....	14-2
Asynchronous Communication Protocol .....	14-3
Data Transfers Between Computer and Interface .....	14-4
Outbound Control Blocks .....	14-4
Inbound Control Blocks .....	14-5
Outbound Data Messages .....	14-5
Inbound Data Messages .....	14-6
Overview of Datacomm Programming .....	14-7
Establishing the Connection .....	14-8
Determining Protocol and Link Operating Parameters .....	14-8
Datacomm Configuration for BASIC/UX .....	14-9
Resetting the Datacomm Interface .....	14-11
Datacomm Options for Async Communications .....	14-12
Connecting to the Line .....	14-15
Connection Procedure for Hayes-Compatible Modems .....	14-16
Datacomm Error Recovery .....	14-18
Datacomm Error Detection and Program Recovery .....	14-19
Cable and Adapter Options and Functions .....	14-20
DTE and DCE Cable Options .....	14-20
Optional Circuit Driver/Receiver Functions .....	14-21
RS-232C/CCITT V24 .....	14-23
The HP 98642 4-Channel Multiplexer .....	14-25
Specifics on the HP 98642 4-Channel Multiplexer .....	14-25
Using the HP 98642 4-Channel Multiplexer .....	14-25
Keywords Used by the HP 98642 4-Channel Multiplexer .....	14-26
HP 98628 and HP 98642 Datacomm Interface Status and Control Registers .....	14-28

## **Chapter 15: The GPIO Interface**

Introduction .....	15-1
Interface Description .....	15-2
Interface Configuration .....	15-4
Interface Select Code .....	15-5
Hardware Interrupt Priority .....	15-5
Data Logic Sense .....	15-5
Data Handshake Methods .....	15-5
Interface Reset .....	15-17



Outputs and Enters through the GPIO .....	15-18
ASCII and Internal Representations .....	15-18
Using a GPIO Interface in the HP-UX Environment .....	15-24
GPIO Timeouts .....	15-26
Using Alternate Data Representations .....	15-28
BCD Representation .....	15-28
Character Conversions .....	15-31
GPIO Interrupts .....	15-32
Types of Interrupt Events .....	15-32
Setting Up and Enabling Events .....	15-32
Interrupt Service Routines .....	15-34
Designing Your Own Transfers .....	15-37
Full Handshake Transfer .....	15-38
Interrupt Transfers .....	15-39
Using the Special-Purpose Lines .....	15-41
Driving the Control Output Lines .....	15-41
Interrogating the Status Input Lines .....	15-42
Using the PSTS Line .....	15-43
Summary of GPIO STATUS and CONTROL Registers .....	15-44
Summary of GPIO READIO and WRITEIO Registers .....	15-47
GPIO READIO Registers .....	15-47
GPIO WRITEIO Registers .....	15-49

## **Chapter 16: HP-HIL Interface**

The Interface to HP-HIL Devices .....	16-1
Preview of HP-HIL Devices .....	16-3
Communicating through the HP-HIL Interface .....	16-4
Supported HP-HIL Devices .....	16-6
Selecting HP-HIL Devices .....	16-6
Identifying All Devices on the HP-HIL Link .....	16-8
Explanation of the HIL_ID Program .....	16-10
HP-HIL Devices .....	16-22
Communicating with HP-HIL Devices .....	16-31
HP-HIL Device Characteristics .....	16-31
ID Module .....	16-32
Function Box .....	16-35
Using a Touchscreen .....	16-46
Using a Bar Code Reader .....	16-51
Interaction Among Multiple HP-HIL Devices .....	16-55

## **Chapter 17: Using BASIC/UX in the X Window System**

Windowing Operations with BASIC/UX .....	17-1
Creating Windows .....	17-2
Listing Windows .....	17-4
Removing Windows .....	17-6
Moving Windows .....	17-7
Outputting Graphics to a Window .....	17-8
Clearing the Contents of Windows .....	17-10
Raising and Lowering a BASIC/UX Window in the Window Stack ....	17-10
Copying Data Between Windows .....	17-12

## **Appendix A: HP-HIL Appendix**

HP-HIL Command Reference .....	A-2
Identify and Describe (IDD) .....	A-2
Read Register (RRG) .....	A-3
Write Register (WRG) .....	A-3
Report Name (RNM) .....	A-4
Report Status (RST) .....	A-4
Extended Describe (EXD) .....	A-5
Report Security Code (RSC) .....	A-5
Disable Keyswitch Autorepeat (DKA) .....	A-6
Enable Keyswitch Autorepeat (EKA 1,EKA 2) .....	A-6
Prompt 1 thru Prompt 7 (PRM 1 .. PRM 7) .....	A-7
Prompt (PRM) .....	A-7
Acknowledge 1 thru Acknowledge 7 (ACK 1 .. ACK 7) .....	A-8
Acknowledge (ACK) .....	A-8
Device-Dependent Commands (DDC 128 .. 239) .....	A-9
Device ID Byte .....	A-10
Describe Record .....	A-13
Extended Describe Record .....	A-17
Poll Record .....	A-20
Report Security Code Record .....	A-23
Accessible Keycode Definitions .....	A-29





# Table of Contents

---

## Chapter 10: Display Interfaces

Description of Displays .....	10-1
Display Specifics .....	10-2
How the Default Display Device Is Chosen .....	10-2
Overview of Display Features .....	10-3
Clearing the Screen .....	10-5
The Output Area and the Display Line .....	10-5
Determining Screenwidth and Other Attributes .....	10-7
Changing Pen Colors in Display Regions .....	10-8
Making Graphics Dominant .....	10-14
Output to the CRT .....	10-15
Numeric Outputs .....	10-15
String Outputs .....	10-16
Control Characters .....	10-16
Display-Enhancement Characters .....	10-19
The Display Functions Mode .....	10-21
Output-Area Memory .....	10-23
Determining Above-Screen Lines .....	10-23
The PRINT Position .....	10-25
Scrolling the Display .....	10-26
Entering from the CRT .....	10-28
Reading a Screen Line .....	10-28
Reading the Entire Output-Area Memory .....	10-29
Additional CRT Features .....	10-31
The DISP Line .....	10-31
Disabling the Cursor Character .....	10-32
Enabling the Insert Mode .....	10-33
Softkey Labels .....	10-34
Softkey Label Colors .....	10-36
Summary of CRT STATUS and CONTROL Registers .....	10-37



# Display Interfaces

---

This chapter describes programming techniques for sending data to and entering data from display interfaces<sup>1</sup>. Configuring and accessing these devices with I/O statements (OUTPUT, ENTER, STATUS, and CONTROL) is described in this chapter. Many of the I/O concepts and programming techniques presented in preceding chapters (volume 1) of this manual are applied in this chapter. Thus, the display screen is a convenient tool for visually verifying data output before attempting to send it to another device.

---

## Description of Displays

This section provides an overview of the capabilities of display devices available with Series 300 computers. Here are the topics:

- Display Specifics
- How the “default display device” is chosen (in machines with more than one display installed).
- Overview of display features.

---

<sup>1</sup> Using a display for graphics is not described in this chapter. See the *BASIC Graphics Techniques* manual for that information.



## Display Specifics

When CRT or select code 1 is used in this chapter, it is also referring to a BASIC/UX window number in the range of 600 to 699 (unless otherwise indicated).

Displays with *combined* alpha and graphics “planes” (all Series 300 displays).

- Alpha and graphics share the same screen, and both are produced by bit-mapping hardware.
- Alpha and graphics cannot be turned on and off independently (color displays have a special mode in which you can do this).

All Series 300 computer displays have combined alpha and graphics. However, you can configure Series 300 color (multi-plane) displays to use independent alpha and graphics planes. Configuring the color display is discussed in the “Porting to Series 300” chapter of *BASIC Programming Techniques*. Interactions between graphics and alpha planes is described in “Multi-Plane Bit-Mapped Displays” in the “Using Graphics Effectively” chapter of *BASIC Graphics Techniques*.

## How the Default Display Device Is Chosen

Select code 1<sup>1</sup> is always assigned to the “default display device.” However, as previously mentioned, Series 300 computers can have more than one display installed at one time. In such cases, the BASIC system has to choose which display will be the default display device. Here is the order that the system chooses this device:

1. The “internal” bit-mapped display.
  - a. A Series 300 bit-mapped display.
  - b. The HP 98700 Display Controller at “internal” select code 6 (i.e., the “internal/external” switch is set to “internal”).
2. An “external” bit-mapped display.
  - a. An HP 98700 Display Controller at the lowest “external” select code (i.e., the “internal/external” switch is set to “external”, and the select code switches are set to a select code in the range 8 through 31).

---

<sup>1</sup> BASIC provides the CRT function which returns a value of 1.

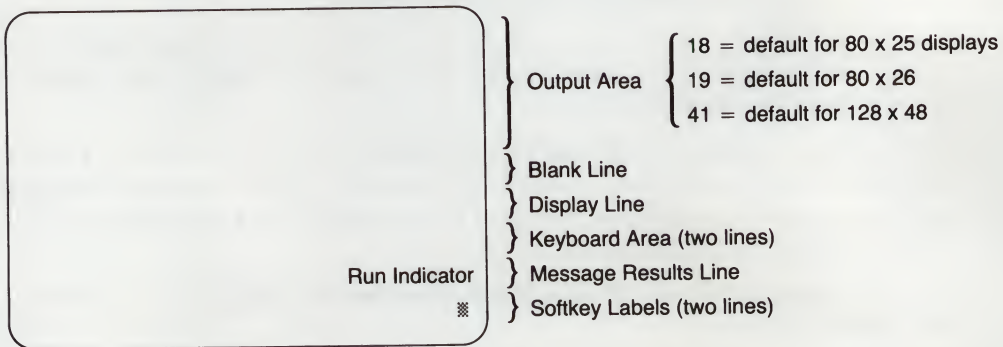
## Overview of Display Features

Even though there are several choices of displays available with Series 300 computers, all have similar alphanumeric display capabilities. Here are the general categories of I/O operations you can perform with the alphanumeric display.

- You can OUTPUT characters to any location on the screen through select code 1 (this applies only to the logical region known as the “output area,” which is described in the next section).
- You can read characters from any location of the output area with the ENTER statement through select code 1. A line-feed character, CHR\$(10), is sent following the last non-blank character on the line. (A simulated HP-IB End-or-Identify, EOI, signal is also sent with the line-feed.)
- You can configure and read the current modes of display operation with CONTROL and STATUS registers.

## Display Regions

The BASIC system logically partitions the alphanumeric display into several regions, each of which has a specific use. Here is a diagram of the regions.



**Figure 10-1. BASIC Display Organization**

Here is a description of each region, from top to bottom of the display.

- The “Output Area” is the logical region on which characters sent to the screen are displayed (characters sent with OUTPUT through select code 1, or with PRINT when PRINTER IS CRT).
- One blank display line separates the output area from other areas.
- One display line is used for output using the DISP statement.
- Two display lines are used for keyboard input and output (input with INPUT, LINPUT, and ENTER through select code 2; OUTPUT through select code 2).
- One display line is used for system “messages” and “results.” These include error messages or results of keyboard computations.

The right end of this line is used by the BASIC system for “annunciators” such as softkey mode (such as **System** or **User 1**), **CAPS** indicator, and system activity indicator (such as **Running**, **Idle**, or **Command**). For more information, see the *Using the BASIC/UX System* manual.

- Two lines at the bottom of the screen are used for softkey labels. (See the subsequent section called “Softkey Labels” for further information.)



While the number of lines in the output area may vary according to display size, BASIC provides all of these regions on all alpha displays.

## Clearing the Screen

Alpha display memory can be cleared using the `CLEAR SCREEN` statement.

```
CLEAR SCREEN
```

It has the same effect as executing:

```
OUTPUT KBD;CHR$(255)&"K";
```

or pressing the Clear display key.

Note that you can type `CLS` and the statement `CLEAR SCREEN` will appear in the program listing. `CLS` acts as a shorthand method of entering the `CLEAR SCREEN` statement.

## The Output Area and the Display Line

There are two separate areas on the CRT used for displaying characters:

- Characters sent to the CRT with `PRINT` and `OUTPUT CRT;...` statements are displayed in the output area.
- Characters sent to the display with the `DISP` statement appear in the display line.

Type in and run the following program to see these areas.

```
10  ALPHA HEIGHT 25    ! or CONTROL 1,13;25
20  FOR K=1 TO 18
30    PRINT "This is line #";K;" in the output area."
40    WAIT .5
50    DISP "This is line #";K;" in the display line."
60    WAIT .5
70  NEXT K
80  END
```

The number of Output Area lines available can vary depending on what display you have.

- Series 300 medium-resolution (bit-mapped alpha) displays provide 19 lines.
- Series 300 high-resolution (bit-mapped alpha) displays provide 41 or 44 lines.

The number of lines in the output area can be altered, within display-size limits, by either of these statements:

ALPHA HEIGHT *Number of lines*

or

CONTROL CRT,13;*Number of lines*

The number of lines specifies an area of the display that begins *at the bottom of the screen*. That is, if you have a default Output Area size of 18 and you execute ALPHA HEIGHT 9, the new Output Area will be the 2 lines at the bottom of the screen (just above the Display Line).

The lower limit of the alpha height for the root window and CRT is 9. The upper limit is 25, 26, 48, or 51 depending on the size of display you have. You can experiment with the ALPHA HEIGHT parameters to find the upper limit for your display. Experiment with different parameters for the ALPHA HEIGHT statement in line 10 in the preceding program. Try integers in the range 9 to 51. Error messages alert you to out-of-range situations.

You should experiment with the loop upper limit in line 20. After the program executes, scroll the displayed text up and down to see what happens. Try values such as 10, 30, 48, 49, 51, and 52.

To determine the current height of the alpha area for a CRT or window number, use STATUS register 13:

```
15 STATUS 1,13;Height
```

To re-establish the default alpha height, omit the "number of lines" parameter in the preceding ALPHA HEIGHT statement:

```
ALPHA HEIGHT
```

Windows created with the `rmb` command (it runs the BASIC interpreter on HP-UX) must have a height of at least 1 for the "`rmb*Geometry`" entry of the `x.defaults` file. Windows with height values less than 10 will have an alpha height of 9, even if the displayed window has fewer lines. This means that output to the windows would be clipped.

## Determining Screenwidth and Other Attributes

A wide variety of displays having different characteristics are available for Series 300 computers. Since all programs are transportable between these computers, a program that uses the display extensively should have the ability to distinguish the characteristics of the display it is using. This BASIC language system provides this capability.

Interface select code 1 is used to access the CRT from BASIC programs. You can also use a window number in the range from 600 to 699. Several registers are associated with this interface which allow the interrogation and control of the CRT/window. For example, STATUS register 13 (discussed previously) returns the current alpha height; STATUS register 9 of the CRT interface returns the screenwidth:

```
STATUS 1,9;Screenwidth
```

STATUS register 19 of the CRT interface returns the maximum alpha mask of the CRT:

```
STATUS 1,19;Max_alpha_mask
```

You can also use `SYSTEM$("CRT ID")` to determine screenwidth (and other display attributes). Executing this function returns a string similar to the following:

```
6: 80HCGB15
```

where:

- 6: is a format indicator for Series 300 computers.
- 80 is the CRT width in characters.
- H indicates that CRT highlights are available (blinking, underlining, etc.).
- C indicates that color is available.
- G indicates that graphics is available.
- B indicates that your display is bit-mapped.
- 15 indicates the highest graphics pen number:
  - 1 if monochrome
  - $2^n - 1$  if bit-mapped ( $n$  = number of planes)



The numeric characters following the bit-mapped indicator represent the CRT “max pen” value (highest graphics pen number available on your display: for instance 4-plane color displays have a max pen equal to 15, 6-plane bit-mapped displays have a max pen equal to 63 or  $2^6-1$ , and so forth).

For more information on `SYSTEM$("CRT ID")` results, read the section covering the `SYSTEM$` function in the *BASIC Language Reference*.

## Changing Pen Colors in Display Regions

This section covers a set of statements which allow you to change the alpha pen colors in five of the display regions on your CRT/window. Here is a list of the BASIC statements which change the alpha pen colors of the display regions:

- `ALPHA PEN pen value` (same as `CONTROL CRT,5; pen value`)
- `KBD LINE PEN pen value` (same as `CONTROL CRT,17; pen value`)
- `KEY LABELS PEN pen value` (same as `CONTROL CRT,16; pen value`)
- `PRINT PEN pen value` (same as `CONTROL CRT,15; pen value`)

The diagram below shows the areas which these statements affect.



**Figure 10-2. Display Regions Affected by Pen Color Statements**

Note that the `KBD LINE PEN` statement sets the color for more than just the Message/Results Line. This statement also sets the color for the “run-light” and for the program lines listed on screen in the edit mode.

The following table should help to clarify the previous diagram of the display regions affected by pen color statements.

**Table 10-1. Display Regions Affected by Pen Color Statements**

Display Region	Written by These Statements	Statements which Affect Pen Colors
Output area	PRINT OUTPUT CAT LIST etc.	ALPHA PEN PRINT PEN
Display line	DISP INPUT (prompt only) LINPUT (prompt only)	ALPHA PEN PRINT PEN
Edit area (replaces Output area and Display line in edit mode)	EDIT	ALPHA PEN KBD LINE PEN
Keyboard area	Keyboard input	ALPHA PEN KBD LINE PEN
Message Results Line	Error messages, keyboard computation results, and annunciators	ALPHA PEN KBD LINE PEN
Softkey Labels	EDIT KEY SET KEY LOAD KEY ON KEY	ALPHA PEN KEY LABELS PEN

### Pen Colors Example

The following program shows the use of all the alpha pen color statements previously mentioned.

```
100 ALPHA PEN 1           ! White (for all display regions).
110 ON KEY 1 LABEL "KeyLabel" GOSUB Sftkey_label
120 GOSUB Show_regions
130 WAIT 3
140 !
150 PRINT PEN 2           ! Red (OUTPUT/PRINT and DISP regions).
160 KBD LINE PEN 3        ! Yellow (Keyboard input line).
170 KEY LABELS PEN 4     ! Green (Softkey labels).
180 WAIT 3
190 CLEAR LINE
200 GOSUB Show_regions
210 STOP
220 !
230 Show_regions: !
240 PRINT "PRINT/OUTPUT Area"
250 DISP "DISP Line";
260 OUTPUT KBD;"Message/Results Line";CHR$(255)&"E";
270 OUTPUT KBD;RPT$("KBD Line ",10);
280 Sftkey_label:RETURN
290 END
```



The results on the display (through the WAIT 3 statement on **line 180**) are output using an alpha pen color of white.

PRINT/OUTPUT Area

DISP Line

KBD Line KBD Line KBD Line KBD Line KBD Line KBD Line KBD Line KBD Line

KBD Line KBD Line

Message/Results Line

User 1

Idle

KeyLabel Continue RUN SCRATCH LOAD "" LOAD BIN LIST BIN RE-STORE  
"" ""

The preceding program assumes you are using a medium resolution monitor with a Series 300 computer and that you are in the non-color-map mode. The following is an explanation of the program:

**Line 100** selects white as the alpha pen color.

**Line 120** calls a subroutine called **Show\_regions**. This subroutine causes the output of messages to the five CRT display regions. The color of these messages is determined by the alpha pen color statement executed prior to the call to the subroutine. Note that the color may vary from those stated if you are running the program from the X Window environment.

**Line 130** causes the program to wait for 3 seconds before proceeding.

**Line 150** sets the pen color to red for the output area and display line of the CRT (existing text is not affected).

**Line 160** sets the pen color to yellow for the keyboard area and message/results line of the CRT. (Note that the Keyboard lines, annunciators, and "run-light" are updated, the result text is not.)

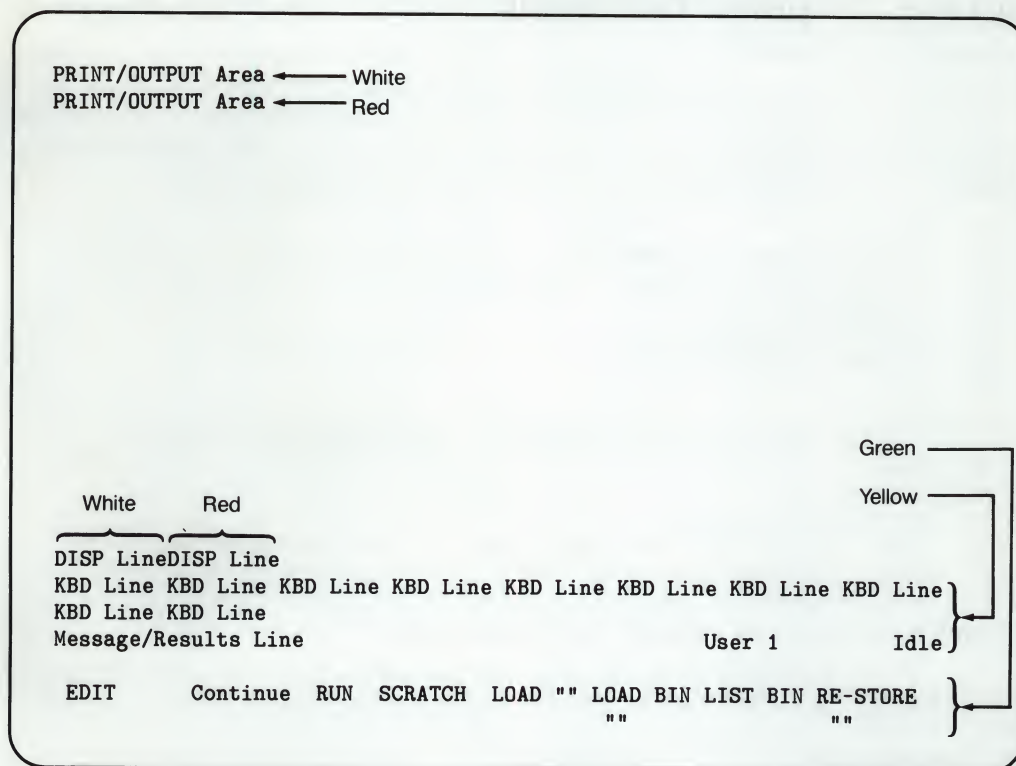
**Line 170** sets the pen color to green for the softkey area of the CRT.

**Line 180** causes the program to wait for 3 seconds before proceeding.

**Line 200** calls the subroutine called **Show\_regions**. This subroutine causes the output of messages to the five CRT display regions. The color of these messages is determined by the pen color statements executed prior to the call to the subroutine.

The remaining program lines are the subroutine called **Show\_regions** which has been previously explained.

The following display shows what happens after the WAIT 3 statement on line 180.



The above display assumes you have a medium-resolution color monitor.

---

## Making Graphics Dominant

Normally, SEPARATE ALPHA sets up the color map to make alpha text dominant over graphics. In this context, dominant means the alpha text appears to be in front of the graphics image (see SEPARATE ALPHA in the *BASIC Language Reference* for more details). The following example shows how to modify this to make graphics images dominant over alpha text (for a system with a 4-plane bit-mapped display).

```
10  DIM A(7,2)
20  SEPARATE ALPHA FROM GRAPHICS
30  PLOTTER IS CRT,"INTERNAL";COLOR MAP
40  GESCAPE CRT,2;A(*)
50  SET PEN 8 INTENSITY A(*)
60  SET PEN 8 INTENSITY 0,1,0
70  !
80  FOR I=0 TO 17
90    PRINT "TEST OF DOMINANT GRAPHICS -- TEST OF DOMINANT GRAPHICS"
100 NEXT I
110 MOVE 30,30
120 RECTANGLE 50,50,FILL
130 DISP "Use the Alpha and Graphics Keys to turn them on and off."
140 END
```

Line **10** creates an array for the first eight color map entries.

Line **20** separates the display into monochrome alpha and color graphics.

Line **30** enables the SET PEN statement.

Line **40** reads the first eight color map entries into the array.

Line **50** duplicates the first eight entries in the next eight color map entries.

Line **60** re-establishes the alpha pen (PEN 8) as green (instead of black).

Lines **80** through **100** fill the output area with text.

Lines **110** through **120** draw a solid box over the bottom center of the output area.

Line **130** prompts the user to experiment with the new relationship between alpha and graphics.

When graphics is on, a white box covers part of the green text so that it cannot be seen. Normally, the green text would be visible over the white box (without the color map changes made in line **50** above).



---

## Output to the CRT

Data can also be sent to the output area by directing OUTPUT statements to interface select code 1 or a window number. The following example uses an I/O path name to direct the data to the CRT; the default data representation used with the CRT is the ASCII representation.

```
100  ASSIGN @Printer TO 1
110  !
120  CLEAR SCREEN
130  FOR Line=1 TO 18
140      OUTPUT @Printer;"The OUTPUT Area"
150  NEXT Line
160  !
170  END
```

### Numeric Outputs

When numbers are output to the CRT/window by the free-field form of the OUTPUT statement, the standard numeric format is used<sup>1</sup>. The following statements show how trailing punctuation within the OUTPUT statement affects the item terminators output after each numeric item.

Examples	Results
OUTPUT 1;123,456	123, 456
OUTPUT 1;-123,456	-123, 456
OUTPUT 1;-123,-456	-123,-456
OUTPUT 1;-123;-456	-123-456
OUTPUT 1;123;456	123 456

*Leading "+" signs are replaced by a space.*

---

<sup>1</sup> "Standard numeric format" is further described in the chapter "Outputting Data".

## String Outputs

Strings are output to the CRT or window in a similar manner with free-field outputs; trailing punctuation in the statement determines whether or not string-item and statement terminators are output. The following examples show how trailing punctuation within the OUTPUT statement affects the output of string-item terminators.

Examples	Results
OUTPUT 1;"One","Two"	One Two
OUTPUT 1;"Three";"Four"	ThreeFour

As with free-field outputs to other devices, a trailing semicolon causes the separator of the item that it follows to be suppressed. In the above case, the carriage-return and line-feed separators which normally follow the output of a string item are suppressed by the semicolon. The next paragraphs describe how the carriage-return and line-feed (control characters) are interpreted by the CRT.

## Control Characters

ASCII characters with codes 0 through 31 are defined to be "control" characters. When one of these characters is sent to a system resource, it is usually interpreted as a **command**, rather than as data. The complete list of control characters and their corresponding codes and definitions is given in the ASCII table in the "Useful Tables" Appendix.

**Four** of these characters are used for controlling the CRT display, and all others are ignored (i.e., are not displayed and cause no special action when received by the CRT). Run the following program and note the results.

```

130 Backspace$=CHR$(8)
140 Line_feed$=CHR$(10)
150 Form_feed$=CHR$(12)
160 Carriage_return$=CHR$(13)
170 !
180 !
190 ASSIGN @Crt TO 1
200 !
210 OUTPUT @Crt;"Back";
220 WAIT 1
230 OUTPUT @Crt;Backspace$;"space"
240 WAIT 1
250 !
260 OUTPUT @Crt;"Line";
270 WAIT 1
280 OUTPUT @Crt;Line_feed$;"feed"
290 WAIT 1
300 !
310 OUTPUT @Crt;"Carriage";
320 WAIT 1
330 OUTPUT @Crt;Carriage_return$;"return"
340 WAIT 1
350 !
360 OUTPUT @Crt;"Form";
370 WAIT 1
380 OUTPUT @Crt;Form_feed$;"feed"
390 DISP "Scroll down to view previous display."
400 !
410 END

```

#### Display Before Scroll

```

feed

```

#### Display After Scroll

```

Backspace
Line
      feed
returnge
Form

feed

```

The following table describes the display functions invoked when the specified control character is sent to the CRT/window (in the "Display functions off" mode). The **print position** is the column and line at which the next character sent to the display will appear.

**Table 10-2. Control-Character Functions on the CRT**

Character	ASCII Code	Defined Action
Bell	7	Causes beeper to output the standard tone; no display action is invoked.
Backspace (BS)	8	If the print position was not in column 1, it is moved "back" one character position; if it was in the first column, no action is invoked.
Line-feed (LF)	10	Moves the print position "down" one line.
Form-feed (FF)	12	Prints two blank lines, scrolls the screen "up" as far as possible, and places the print position at column 1 of the second, printed blank line.
Carriage-return (CR)	13	Causes the print position to be moved to the beginning (first column) of the current screen line.
All other control characters	—	Ignored.

Here is another short example program. Type it in and watch the execution. Notice how you can use CONTROL statements and selected registers to control the CRT display.

```

100  CONTROL 1,1;10          ! Go to 10th line
110  PRINT "Let's back up a little." ! Print a sentence
120  WAIT .6                 ! Time to see it.
130  FOR K=24 TO 1 STEP -1    ! Loop for backspacing.
140      CONTROL 1,1;10      ! Stay on 10th line. Could do other ways.
150      CONTROL 1,0;K       ! Move cursor back one space.
160      PRINT CHR$(8);" ";  ! Issue backspace.
170      WAIT .3             ! Lets you see backspacing.
180  NEXT K                  ! Continue loop.
190  END

```



## Display-Enhancement Characters

Characters with codes 128 through 159 are a second set of “control” characters. Some of these characters are used to implement display-enhancements. The others are ignored.

### Monochrome Enhancement Characters

Some displays have the ability to display underlined, blinking, and inverse-video characters. Both the Output Area and the Display Line have these abilities.

**Table 10-3. Monochrome Display-Enhancement Characters**

Character Code	Action Resulting from Displaying the Character
128	All enhancements off.
129	Inverse mode on.
130	Blinking mode on. *
131	Inverse and Blinking modes on. *
132	Underline mode on.
133	Underline and Inverse modes on.
134	Underline and Blinking modes on. *
135	Underline, Inverse, and Blinking modes on. *

\* *No blinking on bit-mapped alpha displays.*

When one of these characters is sent to the CRT/window, it turns on the corresponding enhancement(s). All subsequent characters on the CRT/window are also displayed in the specified enhancement mode; if only a few characters are to be enhanced, a **CHR\$(128)** must be sent to the display after the last character to be enhanced, which turns off **all** enhancements.

From the preceding table, you may have deduced that certain bits within the character bytes turn on these display modes. The following bit pattern and individual bits control these features.

**Most Significant Bit**

**Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	0	0	0	Underline On	Blinking On	Inverse On
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Notice that the upper five bits (7 through 3) must be in the pattern shown (numeric value = 128). Thus, adding the values 4, 2, or 1 enable the Underline, Blinking, and Inverse features. Several examples of using these enhancements are shown in the following program.

```
100  PRINTER IS 1
110  Off=128
120  Underline=4
130  Blinking=2
140  Inverse=1
150  !
160  PRINT CHR$(Off);"Normal"
170  PRINT
180  PRINT CHR$(128+Inverse);"Inverse"
190  PRINT "carries over onto"
200  PRINT "subsequent lines"
210  PRINT
220  PRINT CHR$(128+Underline);"Underline"
230  PRINT "also remains on until turned off"
240  PRINT
250  PRINT CHR$(128+Blinking);"Blinking"
260  PRINT "is the same"
270  PRINT
280  PRINT CHR$(Off);"Back to normal"
290  PRINT
300  END
```

### Color Enhancements

Color displays recognize 8 additional control codes for selecting the first 8 alpha pen colors. Both the Output Area and the Display Line have this ability.

Table 10-4. Color Enhancements

Character Code	Model 236C Display	Bit-mapped Alpha Display
136	White	PEN 1
137	Red	PEN 2
138	Yellow	PEN 3
139	Green	PEN 4
140	Cyan	PEN 5
141	Blue	PEN 6
142	Magenta	PEN 7
143	Black	PEN 0

When using BASIC/UX from the X Window environment, the colors will correspond to the first eight colors of the X Window color map.

These same features can also be placed in strings by using the `[f7]` key (**Any Char**) while typing in string literals. Keep in mind that, even though these characters are not shown on the screen, they are counted in the length of the string. Dimension string variables accordingly.

## The Display Functions Mode

The preceding program showed the control characters which are defined to invoke a special display function when sent to some CRTs or windows. To display **all** control characters sent to the CRT/window, rather than have the CRT/window interpret them as commands, turn the Display Functions mode **on** by pressing the **Display Fctns** key (`[f6]` in the System menu). Repeatedly pressing this key toggles this display mode between "on" and "off". The same thing can be accomplished programmatically using the statement `DISPLAY FUNCTIONS ON/OFF`. Using the CRT/window with `DISPLAY FUNCTIONS ON` is very useful when you need to see exactly which control characters have been output. An asterisk is visible in the Display Functions softkey label when the Display Functions mode is on.

Except for the carriage-return character, all subsequent control characters sent to the display (while in this mode) do not invoke their defined function, but are **only displayed**. The carriage-return is **both** displayed **and** causes the print position to move to the beginning of the next line (both CR and LF functions invoked).

The `DISPLAY FUNCTIONS` mode can also be enabled from BASIC programs with the use of the `CONTROL` statement. The following program shows how this is accomplished. Notice that the carriage-return invokes both carriage-return and line-feed functions.

```
100  DISPLAY FUNCTIONS ON    ! CONTROL CRT,4;1
110  !
120  ! First send with default CR/LF sequence.
130  OUTPUT 1;"DISPLAY FUNCTIONS ON"
140  !
150  ! Then suppress the CR/LF (with ";").
160  OUTPUT 1;CHR$(12);
170  END
```



Notice that the **DISPLAY FUNCTIONS ON** message normally displayed when the key is pressed is not automatically displayed when the mode is changed by one of these statements; instead, the program must display the message, if so desired.

The following program uses the CRT mapping register (**lines 150 and 270**) and the **DISPLAY FUNCTIONS** statements (**lines 160, 210, 230, and 260**) to display the logical and physical CRT character sets.

```
100  ! Output available character sets to CRT
110  PRINT "If you want to see the ROM contents, THEN ENTER any character"
120  PRINT "ELSE, to see the legitimate character set, ENTER a null string."
130  INPUT A$
140  PRINT CHR$(12)          ! Issue a form feed.
150  CONTROL 1,11;LEN(A$) ! Identify which character set.
160  DISPLAY FUNCTIONS ON ! Set display functions mode.
170  ! Provide loop to display character set.
180  FOR I=0 TO 255
190      PRINT USING "#,DDD,X,A,X";I,CHR$(I)
200      IF NOT (I MOD 13) THEN
210          DISPLAY FUNCTIONS OFF
220          PRINT
230          DISPLAY FUNCTIONS ON
240      END IF
250  NEXT I
260  DISPLAY FUNCTIONS OFF
270  CONTROL 1,11;0
280  PRINT
290  END
```

The logical and physical CRT character sets are the same on all bit-mapped displays when the "character-set select" switch is set to the ASCII/ROMAN 8 position.

Access **DISPLAY FUNCTIONS** functionality from windows other than the root window with:

```
CONTROL 699,4;1
```

where **699** is the device selector for window **699**.



---

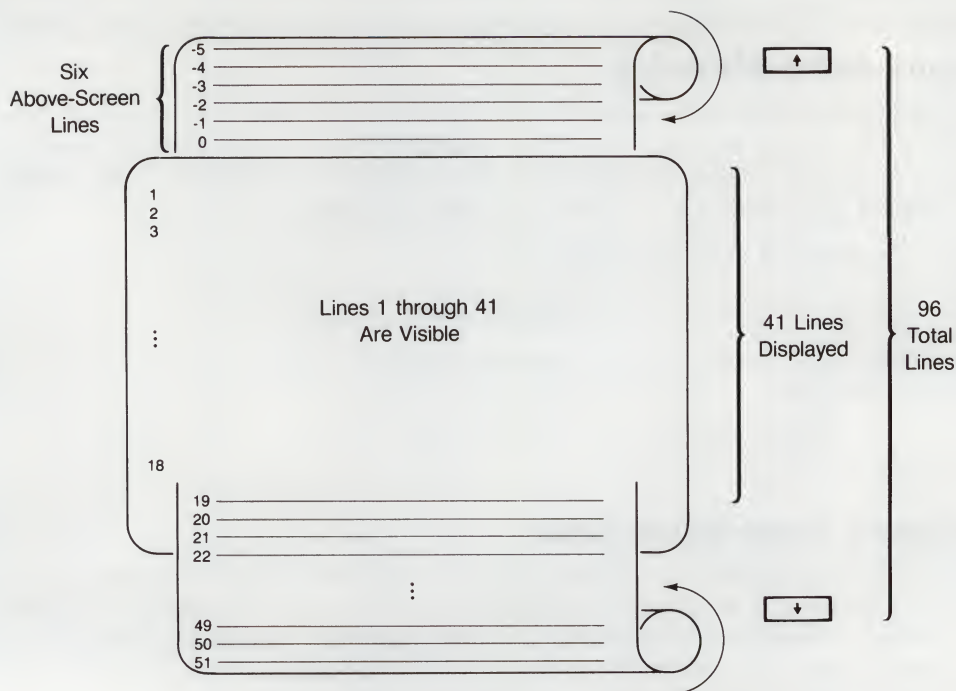
## Output-Area Memory

In addition to the visible display lines in the output area, there may be additional lines available within output-area memory. These additional lines of display memory can be viewed by running the following program and then scrolling the display down. The **visible** lines of output area memory will hereafter be called the **screen**.

```
100  ! Example to show scrolling.
110  !
120  PRINTER IS 1          ! PRINT on CRT.
130  !
140  FOR Line=1 TO 48      ! Write 48 lines.
150    PRINT Line
160  NEXT Line
170  ! Now scroll manually.
180  END
```

### Determining Above-Screen Lines

Scrolling the display up and down allows you to view different portions of the lines within output-area memory. If the display is scrolled **down** as far as possible, there are no lines “above screen”. Similarly, if the display is scrolled **up** as far as possible, there are no lines “below screen”. The following drawing illustrates the screen of an HP 98782A color monitor with 6 lines above the screen.



**Figure 10-3. Line Positions of the Output Area**

The values could vary if you have a different CRT/window.

The number of lines that are above screen can be determined from BASIC programs by reading STATUS register 3 of interface select code 1. The returned value is the number of lines currently **above screen**.

If the screen has just been cleared (Clear display), the following program displays:

**0 lines above screen.**

Running the program a second time displays:

**41 lines above screen**

Subsequently running the program produces similar results, until the following message is displayed continually (on a 48-line CRT):

```
96 lines above screen

100  FOR Line=1 TO 41
110      OUTPUT 1;Line
120  NEXT Line
130  !
140  STATUS 1,3;Lines_above
150  DISP Lines_above;"  lines above screen"
160  END
```

The final value will be different for different displays.

## The PRINT Position

All of the characters in output-area memory can be addressed individually by the character's screen column and line. The character in the upper left corner of the screen is in column 1 and line 1, and the character in the lower right corner varies depending on what monitor you have. The addresses of the characters "off screen" are limited by the number of lines currently above screen.

The screen addresses (both column and line) at which a subsequent character sent to the display will appear on the screen are known as the **print position**. The current print position is automatically changed as characters are output to the display. For instance, the print-position column is incremented each time a character is sent; when the last character is sent to a line, the print-position column is reset to 1 and the print-position line is incremented, sending the next character to the next line. The following program shows how the print-position line varies during output to the CRT/window.

```
100  FOR Line=1 TO 48
110      OUTPUT 1;Line
120      STATUS 1,1;Print_line
130      DISP "Print-position line = ";Print_line
140      IF Line<25 THEN WAIT .2
150  NEXT Line
160  !
170  END
```

Notice that **the print-position line is always relative to the first line of the current screen.** This accounts for the print position (read with STATUS) remaining at a value of 19 while the 19th through 48th lines are being printed. When the print position is **off screen**, the display is scrolled (when it receives a character) so that the character appears on the screen. When the display is finished scrolling, all line addresses are again relative to the **new** top screen line. The next section describes using this feature to scroll the display from the program.

## Scrolling the Display

A program can scroll the display up and down by changing the print position to a location **off screen** and then outputting character(s) to the CRT/window. In order to **scroll up**, values greater than the number of lines in the output area must be written to register 1. Assuming an alpha height of 25, this number is 18. If the screen is to be scrolled up 4 lines, the following statements can be used. In this case, the OUTPUT statement outputs the "Null" control character so that no characters will be overwritten.

```
100  CONTROL 1,1;18+4    ! Move print position off screen;
110  OUTPUT 1;CHR$(0);    ! scrolling takes place when next
120                                ! character sent to the CRT.
```

The screen is not scrolled up until the OUTPUT statement actually writes to the CRT/window at the current print position (even though, in this case, no visible character is actually output to the display).

In order to **scroll down**, a non-positive number must be written into register 1. For instance, to scroll down one line, a 0 would be written into register 1. Again, the display is not actually scrolled until an OUTPUT (or PRINT) to the CRT is executed.

The only **restriction** on the value of the line number is that it must not attempt to scroll the screen down **past** the first line of output-area memory. In other words, to scroll down as far as possible, the following value would be used; using smaller values results in an error.

Top line's address = - (number of lines above screen) + 1

Thus, if no lines are above screen, the top line's address is 1.



An example of scrolling down "as far as possible" is shown in the following program.

```
100  FOR Line=1 TO 48
110      OUTPUT 1;Line
120  NEXT Line
130  !
140  STATUS 1,1;Line_pos
150  DISP "Print-position line =";Line_pos;" after OUTPUT,"
160  WAIT 2
170  !
180  STATUS 1,3;Lines_above  ! Find # lines above screen.
190  DISP "and";Lines_above;" lines are above screen"
200  WAIT 3
210  !
220  CONTROL 1,1;-Lines_above+1  ! Change line-pos.
230  OUTPUT 1;"Line 1"          ! Scroll made when 1st.
240                                ! character is sent.
250  !
260  STATUS 1,3;Lines_above
270  DISP "Now, number of lines above screen =";Lines_above
280  END
```

---

## Entering from the CRT

Data is entered from the CRT or a window beginning at the current print position. As characters are read from the screen (from left to right), the print position is updated. When the ENTER statement attempts to read past the last non-blank character on a line, the CRT/window driver sends a line-feed character accompanied by a (simulated) EOI signal, and the print position is advanced to the beginning of the next line. Display-enhancement characters, CHR\$(128) through CHR\$(143), cannot be entered from the CRT/window memory<sup>1</sup>.

### Reading a Screen Line

The following program uses the line-feed accompanied by EOI to terminate entry into a string variable. Since the free-field ENTER statement is used, only one line can be read because of the EOI sent with the line-feed character.

```
100  CONTROL 1;5,8          ! Move print position to
110                               ! 5th column of line 8,
120  OUTPUT 1;"ABCDEFGH"     ! then OUTPUT (with CR/LF).
130  !
140  OUTPUT 1;"IJKLMNOP"     " ! OUTPUT to line 9 with
150                               ! trailing spaces.
160                               !
170  CONTROL 1,1;8          ! Move print position back
180                               ! to 1st column of line 8.
190                               !
200  ENTER 1;Line_8$
210  DISP LEN(Line_8$);"characters read from line 8"
220  WAIT 2
230  !
240  ENTER 1;Line_9$
250  DISP LEN(Line_9$);"characters read from line 9"
260  END
```

This feature of the CRT/window is very useful when simulating entry from the HP-IB interface; however, keep in mind that **no spaces can be read after the last visible character at the end of each line**. Notice in the preceding example that the leading space characters in a string which are sent to the display were read by ENTER and trailing space characters sent to the display were **not** read back by the ENTER statement. These trailing characters are treated as "blanks" by the CRT, which sends the line-feed with EOI when the ENTER statement attempts to read the first one.

---

<sup>1</sup> When these characters are shown on the screen (because they were displayed while DISPLAY FUNCTIONS was on), they can be read with ENTER statement. However, if they are instead "executed" (because DISPLAY FUNCTIONS is off), they are not read with ENTER.

## Reading the Entire Output-Area Memory

In order to read all lines within output-area memory, an ENTER statement that uses an image must be used to prevent the EOI signal from terminating the statement prematurely (since the EOI signal acts as an **item** terminator during ENTER-USING-image statements which contain no "%" image specifiers). The following program shows the entire contents of output-area memory being read.

```
100  OPTION BASE 1
110  DIM Memory$(48)[50]           ! To read 48 lines.
120  !
130  FOR Screen_line=1 TO 48
140      OUTPUT 1;"Line";Screen_line
150  NEXT Screen_line
160  WAIT 1
170  !
180  STATUS 1,3;Lines_above
190  CONTROL 1;1,-Lines_above+1    ! Scroll to read
200  ENTER 1 USING "K";Memory$(*) ! entire memory.
210  !
220  FOR Screen_line=1 TO 48      ! Display all lines;
230      PRINT Memory$(Screen_line);" "; ! no CR/LF.
240  NEXT Screen_line
250  END
```

## Final Display

```
Line 36
Line 37
Line 38
Line 39
Line 40
Line 41
Line 42
Line 43
Line 44
Line 45
Line 46
Line 47
Line 48
Line 1 Line 2 Line 3 Line 4 Line 5 Line 6 Line 7 L
ine 8 Line 9 Line 10 Line 11 Line 12 Line 13 Line
14 Line 15 Line 16 Line 17 Line 18 Line 19 Line 20
Line 21 Line 22 Line 23 Line 24 Line 25 Line 26 L
ine 27 Line 28 Line 29 Line 30 Line 31 Line 32 Lin
e 33 Line 34 Line 35 Line 36 Line 37 Line 38 Line
39 Line 40 Line 41 Line 42 Line 43 Line 44 Line 45
Line 46 Line 47 Line 48
```

Notice that the print position was moved to the top line before attempting to read memory contents, since the ENTER statement reads characters beginning at the print position. If the print position is not at the "top line" of memory before attempting to read all 57 lines, the lines above screen will not be read. However, the statement executes with no errors, because the CRT sends line-feeds (with EOI) for each line that does not really exist "below screen". For instance, if the print position is at line 10 when the ENTER begins, only the last 47 lines of output-area memory will be read (and placed into the first 47 elements of Memory\$). When the ENTER statement attempts to fill the last ten elements of Memory\$, the CRT sends only line-feeds accompanied by EOI because the print position is past the last non-blank character.



---

## Additional CRT Features

This section describes the remainder of features of the CRT/window display controllable by BASIC programs. **Interrupt and timeout events are not available with the CRT interface.**

### The DISP Line

BASIC programs can output characters to the DISP Line with the DISP statement, as described in the *BASIC Language Reference*. As with the output-area's print position, the position (column) within the DISP line at which subsequent characters will appear can be read and changed explicitly by BASIC programs. This **DISP-line position** can be read and changed with STATUS register 8 and CONTROL register 8 (of interface select code 1), respectively. Note that the CONTROL register 8 statement can be replaced with the DISP TAB statement as shown in the following program. However, the DISP TAB statement and the CONTROL register 8 statement are slightly different. For example, if there were characters in the first 45 columns of the DISP Line, those characters would be blanked when the DISP TAB statement is executed. The CONTROL register 8 statement allows the characters to remain, and as the FOR loop decrements its count the characters in columns 1 through 45 would not be blanked.

```
100  FOR Disp_pos=46 TO 1 STEP -1
110      DISP TAB(Disp_pos),"HELLO"  ! or 110  CONTROL 1,8;Disp_pos
120                                     !      120  DISP "HELLO"
130                                     !
140      WAIT .2
150  NEXT Disp_pos
160  END
```

Keep in mind that if trailing carriage-return and line-feed characters are output to the DISP line, the carriage-return returns the DISP-line position to column 1. A subsequent DISP statement clears the entire line. However, if these trailing characters are suppressed, the DISP-line position is left unchanged. Run the following program to see these effects.

```
100  PRINT "First with trailing CR/LF,"
110  DISP "HI"
120  WAIT .5
130  DISP " THERE"
140  WAIT 1
150  !
160  PRINT "then with no CR/LF."
170  DISP "HI";
180  WAIT .5
190  DISP " THERE"
200  END
```

Also notice that if a DISP attempts to send characters to the DISP line so that any character will be past the last column (50, 80, or 128 depending on your CRT/window), the entire line is shifted left so that all of the new characters will be displayed (i.e., so that the last character written will end up in the last column).

```
100  A$=SYSTEM$("CRT ID")
110  X=VAL(A$[3])-10 ! Screen width minus 10.
120  DISP TAB(X),"CHARACTERS"; ! No CR/LF.
130  WAIT 1
140  DISP " SHIFTED LEFT"
150  !
160  END
```

The display-enhancement characters produce the same effects in the DISP Line as in the OUTPUT Area.

### Changing Pen Colors

The pen color of the OUTPUT Area and DISP Line of the CRT/window can be changed using either the ALPHA PEN or PRINT PEN statement followed by a *pen value*. Information for both of these commands can be found in the preceding section of this chapter entitled "Changing Pen Colors in Display Regions."

### Disabling the Cursor Character

BASIC programs even have control over whether any cursor is displayed (during all computer modes, such as during EDITS and other keyboard-entry modes). The cursor is **disabled** with the following statement.

```
CONTROL 1,10;0
```

Any **non-zero** value written to this register **re-enables** the cursor to be displayed. Resetting the computer also re-enables the cursor being displayed.

```
CONTROL 1,10;1
```

## Enabling the Insert Mode

The insert mode of the keyboard area can be enabled and disabled with STATUS and CONTROL statements. If any **non-zero** numeric value is written to register 2, the insert mode is **enabled**. All subsequent characters typed into this area are "inserted" between the cursor and the character to its immediate left, and characters to its right are shifted appropriately.

The following program turns insert mode on for approximately five seconds. During this time, use the arrow keys to move the cursor left and right while typing in characters from the keyboard.

```
100  Insert_mode=1
110  CONTROL 1,2;Insert_mode
120  !
130  DISP "Insert mode is now being used."
140  BEEP 200,.2
150  WAIT 5
160  !
170  Insert_mode=0
180  CONTROL 1,2;Insert_mode
190  DISP "Now the mode has changed to overwrite."
200  BEEP 100,.2
210  WAIT 5
220  !
230  BEEP 50,.2
240  DISP "Program ended."
250  END
```



## Softkey Labels

Softkeys can be defined as typing-aid keys or as keys that initiate program (ON KEY) branches. In any usage, two display lines (near the bottom of the CRT/window) can be used for key labels. The topic of typing-aid keys is discussed in the *Using the BASIC/UX System* manual. The topic of using softkeys to initiate program branches is discussed in the chapter entitled "Program Structure and Flow" found in the *BASIC Programming Techniques* manual.

Softkey labels can be turned off and on by writing to CRT or window CONTROL Register 12 or using the KEY LABELS ON/OFF statement. The values written to the register have the following effects:

**Table 10-5. Turning Softkey Labels Off/On**

Value of CRT Register 12	Effect on Key Labels
0	Typing-aid key labels are <i>displayed until the program is run</i> , at which time they are turned off (until at least one ON KEY is executed). Annunciators, if present, stay on. System menu softkeys are displayed even when a program is running. (This is the default for systems with an HP 98203A/B/C keyboard.)
1	Typing-aid and softkey labels are <i>not displayed at any time</i> .
2	Typing-aid and softkey labels are <i>displayed at all times</i> . (This is the default for systems with an ITF keyboard.)

The default value of this register is 2 for BASIC/UX, since it uses an ITF keyboard. The default is restored at power-on and when SCRATCH A is executed. The register's current contents can be determined by reading STATUS Register 12.



Here is an illustrative program which cycles through Register 12 using the values 0, 1, and 2. Note that loading the binary CRTX allows you to use the statements KEY LABELS ON and KEY LABELS OFF in place of the CONTROL register 12 statements.

```

100  ! Toggle key displays
110  PRINT "Softkey labels are toggled."
120  WAIT 2
130  ! Set up toggle loop.
140  FOR J=1 TO 3
150      FOR Toggle=0 TO 2 ! 1= KEY LABELS ON, and 2= KEY LABELS OFF
160          CLEAR SCREEN
170          PRINT "Flag value =";Toggle
180          CONTROL 1,12;Toggle ! KEY LABELS ON and OFF.
190          WAIT 1.5
200      NEXT Toggle
210  NEXT J
220  !
230  STOP
240  END

```

Try running the program with one of the three User menus and then with the System menu.

You can use CONTROL register 2 (of select code 2) to cycle the menus: 0=System, 1=User 1, 2=User 2, and 3=User 3. For example:

```

CONTROL 2,2;3
or
USER 3 KEYS

```

displays the menu for User 3 softkeys. Another method of bringing the System menu and User menus up is to use the:

- SYSTEM KEYS statement for the System menu.
- USER *menu number* KEYS statement with the appropriate user *menu number* for the three User menus.

CONTROL register 14 (of select code 2) can be used to set softkey bases 0 or 1 (i.e., **f1** is KEY 0, **f2** is KEY 1, etc.) The default is 0, which means the softkeys start at 1. Changing the flag value to 1 starts the softkeys at 0. You might need to deal with this because the softkeys on HP 98203 Keyboards are labeled from 0 to 9, and from 1 to 8 on ITF Keyboards.

Note that you can draw a solid line between the two lines of key labels (on machines with ITF keyboards). Putting a CHR\$(132) as the first character<sup>1</sup> of the label causes the line to be drawn. An example of this technique is the System key labeled **Clr Tab** (above the line) and **Set Tab** (below the line). See the *Using the BASIC/UX System* manual for examples.

## Softkey Label Colors

Softkey pen color changes are made using the KEY LABELS PEN statement followed by the *pen value*. For a detailed description of *pen values* and the use of this statement, read the preceding section in this chapter entitled “Changing Pen Colors in Display Regions.”

---

<sup>1</sup> Third character if the first two are “inverse-video K” (character code 255) and “#”, which represents a Clear line key.

---

## Summary of CRT STATUS and CONTROL Registers

<b>STATUS Register 0</b>	Current print position (column)
<b>CONTROL Register 0</b>	Set print position (column). See also TAB and TABXY.
<b>STATUS Register 1</b>	Current print position (line)
<b>CONTROL Register 1</b>	Set print position (line). See also TABXY.
<b>STATUS Register 2</b>	Insert-character mode
<b>CONTROL Register 2</b>	Set insert character mode if non-0 <sup>1</sup>
<b>STATUS Register 3</b>	Number of lines "above screen".
<b>CONTROL Register 3</b>	Undefined
<b>STATUS Register 4</b>	Display functions mode
<b>CONTROL Register 4</b>	Set display functions mode if non-0. To perform the same function, use the statement DISPLAY FUNCTIONS ON/OFF.

---

<sup>1</sup> Error 713 is given if a window number is specified instead of a select code.

**STATUS Register 5** Returns the CRT alpha color value set (or default). This does not reflect changes due to printing  $\text{CHR}\$(x)$ , where  $136 \leq x \leq 143$ .

**CONTROL Register 5** Set default alpha color:  
For Alpha Displays:

Value	Result
< 16	The number is evaluated MOD 8 and resulting values produce the following: 0 — black 1 — white 2 — red 3 — yellow 4 — green 5 — cyan 6 — blue 7 — magenta
16 to 135	Ignored
136	White
137	Red
138	Yellow
139	Green
140	Cyan
141	Blue
142	Magenta
143	Black
144 to 255	Ignored

For Bit-Mapped Displays:

Values 0 thru 255 which correspond to the graphics pens. The values are treated as  $\text{MOD } 2^n$  where  $n$  is the number of display planes.



CONTROL CRT,5;*n* sets the values of the CRT registers 15, 16, and 17, but the converse is not true. That is, STATUS CRT,5 may not accurately reflect the CRT state if CONTROL 15, 16, and/or 17 have been executed. Note that to perform the same function as CONTROL CRT,5;*n*, you can use the ALPHA PEN statement.

<b>STATUS Register 6</b>	ALPHA ON flag <sup>1</sup>
<b>CONTROL Register 6</b>	Undefined <sup>1</sup>
<b>STATUS Register 7</b>	GRAPHICS ON flag <sup>1</sup>
<b>CONTROL Register 7</b>	Undefined <sup>1</sup>
<b>STATUS Register 8</b>	Display line position <sup>1</sup> (column)
<b>CONTROL Register 8</b>	Set display line position <sup>1</sup> (column). See also TAB.
<b>STATUS Register 9</b>	Screenwidth (number of characters). Also available in the SYSTEM\$("CRT ID") function result.
<b>CONTROL Register 9</b>	Undefined
<b>STATUS Register 10</b>	Cursor-enable flag <sup>1</sup>
<b>CONTROL Register 10</b>	Cursor-enable: <sup>1</sup> 0=invisible cursor. non-0=cursor visible.
<b>STATUS Register 11</b>	CRT character mapping flag
<b>CONTROL Register 11</b>	Disable CRT character mapping (if non-0)
<b>STATUS Register 12</b>	Key labels display mode. <sup>1</sup>
<b>CONTROL Register 12</b>	Set key labels display mode: <sup>1</sup> 0 = typing-aid key labels displayed unless program is running. 1 = key labels always off (or use KEY LABELS OFF). 2 = key labels displayed at all times (or use KEY LABELS ON).

<sup>1</sup> Error 713 is given if a window number is specified instead of a select code.

- STATUS Register 13** CRT height (number of lines to be used for alpha display).
- CONTROL Register 13** Set CRT height (must be  $\geq 9$ ). Alternately use the ALPHA HEIGHT statement.
- STATUS Register 15** Return the value set (or the default) for the color in the PRINT/DISP area. This does not reflect changes due to printing  $\text{CHR}\$(x)$ , where  $136 \leq x \leq 143$ .
- CONTROL Register 15** Set PRINT/DISP color (or use the PRINT PEN statement). Similar to CRT control register 5 but specific to CRT PRINT/DISP areas; that is, it does not affect the areas covered by CRT registers 16 and 17.
- STATUS Register 16** Return the value set (or the default) for the softkey label color.<sup>1</sup>
- CONTROL Register 16** Set key labels color (or use the KEY LABELS PEN statement).<sup>1</sup> Similar to CRT control register 5 but only affects the softkey labels. Does not affect the areas covered by CRT registers 15 and 17.
- STATUS Register 17** Return the value set (or the default) for the color of the “non-enhance” area. This includes the keyboard entry line, runlight, system message line, annunciators, and edit screen.
- CONTROL Register 17** Set “non-enhance” color (or use the KBD LINE PEN statement). This includes the keyboard entry line, runlight, system message line, annunciators, and edit screen. Similar to CRT control register 5 but does not affect the areas covered by CRT control registers 15 and 16.
- STATUS Register 18** Read the alpha write-enable mask.
- CONTROL Register 18** Set alpha write-enable mask to a bit pattern (or use the SET ALPHA MASK statement). When running BASIC/UX in the X Window environment, this CONTROL register is *not* supported.

<sup>1</sup> Error 713 is given if a window number is specified instead of a select code.

- STATUS Register 19** Returns the maximum value for ALPHA MASK argument.
- CONTROL Register 19** Undefined.
- STATUS Register 20** Read the alpha display-enable mask.<sup>1</sup>
- CONTROL Register 20** Set alpha display-enable mask to a bit pattern (or use the SET DISPLAY MASK statement).<sup>1</sup>
- STATUS Register 21** Return compatibility mode (0 or 1).
- CONTROL Register 21** Used to reset softkey fonts.
- STATUS Register 22** Undefined.
- CONTROL Register 22** Raises a window to the top of the window stack if non-zero; pushes a window to the bottom of the stack if zero.
- STATUS Register 23** Returns terminal compatibility mode.
- CONTROL Register 23** Sets terminal compatibility mode.

---

<sup>1</sup> Error 713 is given if a window number is specified instead of a select code.





# Table of Contents

---

## Chapter 11: The Keyboard Interface

Overview .....	11-1
Keyboard Supported .....	11-1
HP 98203 Keyboard Compatibility Mode .....	11-2
Re-Configuring HIL Devices .....	11-2
Overview of Keyboard Features .....	11-2
ASCII and Non-ASCII Keys .....	11-3
The Shift and Control Keys .....	11-3
Keyboard Operating Modes .....	11-7
The Caps Lock Mode .....	11-7
The Print All Mode .....	11-8
Disabling Scrolling .....	11-8
Modifying the Repeat and Delay Intervals .....	11-9
Entering Data from the Keyboard .....	11-11
Sending the EOI Signal .....	11-13
Sending Data to the Keyboard .....	11-14
Sending Non-ASCII Keystrokes to the Keyboard .....	11-14
Second Byte of Non-ASCII Key Sequences (String) .....	11-16
Closure Keys .....	11-22
Softkeys .....	11-24
Sensing Knob Rotation .....	11-25
Enhanced Keyboard Control .....	11-27
Trapping Keystrokes .....	11-27
Mouse Keys .....	11-30
Softkeys and Knob Rotation .....	11-30
Disabling Interactive Keyboard .....	11-31
Locking Out the Keyboard .....	11-32
Special Considerations .....	11-33
Keyboard Status and Control Registers .....	11-34



# The Keyboard Interface

As with displays, access to the keyboard can be made with OUTPUT, ENTER, CONTROL, and STATUS statements. This chapter describes I/O programming techniques for “interfacing” to the keyboard.

## Overview

### Keyboard Supported

The BASIC/UX system runs on the HP-UX operating system, which supports one type of keyboard: the ITF (Integrated Terminal Family) keyboard.

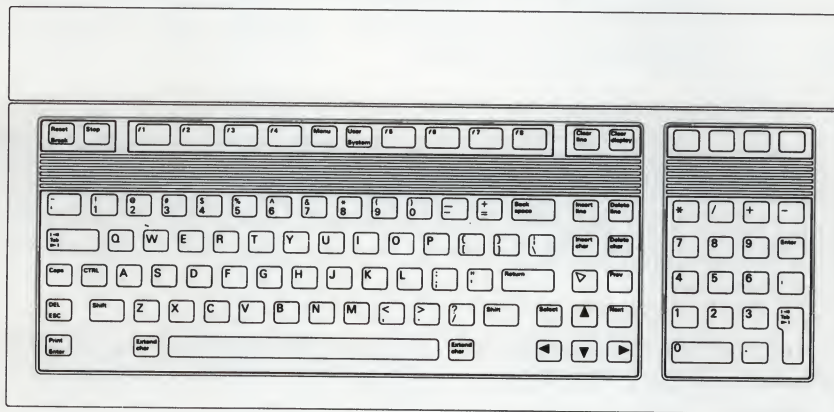


Figure 11-1. An ITF Keyboard

Since BASIC/UX runs on HP-UX, you can also run BASIC/UX from a terminal, which may have a slight variation in the ITF keyboard implementation. Complete descriptions of the BASIC/UX definitions of each key are provided in the “keyboard reference” chapters of the *Using the BASIC/UX System* manual.

## HP 98203 Keyboard Compatibility Mode

There is also a mode of operation, enabled and disabled via keyboard CONTROL register 15 (or the KBD CMODE statement), in which ITF Keyboards can emulate an HP 98203B keyboard; see the “Keyboard Status and Control Register Summary” section at the end of this chapter for values and effects. Details of using this mode are provided in the “Porting to Series 300” chapter of *BASIC Programming Techniques*.

## Re-Configuring HIL Devices

If you add or remove HIL devices while the BASIC system is in the computer, you must re-configure in order for BASIC to properly recognize all devices. Executing SCRATCH A initiates this re-configuration.

## Overview of Keyboard Features

ITF computer keyboards are controlled by their own separate processors, which allows many more capabilities than most other desktop-computer keyboards. These keyboards are devices which reside at select code 2<sup>1</sup>. Here is a brief list of keyboard capabilities:

- You can use the ENTER statement to enter data from the keyboard, and thus simulate devices for debugging purposes.
- You can monitor keys and the “knob” (rotary pulse generator), if present, and enable them to interrupt BASIC programs; the BASIC program can contain a segment of code to read and use this input.
- You can OUTPUT commands to the keyboard, simulating an operator entering them. You can also OUTPUT data to the keyboard which the operator can then edit and send back.

Note, however, that the INTR and TIMEOUT event-initiated branches *cannot* be sensed by the keyboard.

---

<sup>1</sup> BASIC provides the KBD function which returns a value of 2.



## ASCII and Non-ASCII Keys

The keys of an ITF keyboard keyboard can be generally grouped by function into the ASCII and non-ASCII keys.

ASCII (or alphanumeric) keys

all produce an ASCII character when pressed, and include the character entry and numeric keys.

non-ASCII (or non-alphanumeric) keys

do not produce characters but initiate specific actions when pressed; the `[Return]` and `[Back space]` keys are considered to be non-ASCII keys for this reason. Non-ASCII keys also include all program control, editing, cursor control, and system control keys.

## The Shift and Control Keys

The `[Shift]`, `[CTRL]`, and `[Extend char]` keys are not really either type of key because they cannot cause action on their own; instead, they are used only with the other types of keys. Pressing the `[Shift]` key with another key **qualifies** the other keypress, allowing the other key to have a second meaning. For instance, in the “Caps lock off” mode, pressing an alphabetic ASCII key generates a lowercase alphabetic character. Pressing the `[Shift]` key **simultaneously** with an alphabetic key in the “Caps lock off” mode generates an uppercase character. The `[Shift]` key is used similarly with the non-ASCII keys, allowing many of those keys to have a second function.

The `[Extend char]` key is held down while you press other keys from the main typewriter section to generate the rest of the available 256 ASCII characters. It also has a special use with the softkeys when in keyboard compatibility mode, see the chapter entitled “Porting to Series 300” found in the *BASIC Programming Techniques* manual.

The `[CTRL]` (Control) key is also used to further qualify **both** ASCII and non-ASCII keypresses. Pressing the `[CTRL]` key simultaneously with an ASCII key generates an ASCII control character in the display, and is often faster than using the **Any char** (`[F7]`) softkey. The following table shows how to generate control characters by simultaneously pressing the `[CTRL]` key and a key as listed. This is particularly useful when you need to include a control character in a string.

Table 11-1. Generating Control Characters with CTRL and ASCII Keys

Character Code	ASCII Character	Character's Description	Key(s) Pressed with CTRL	Character on CRT
0	NUL	Null	(space bar)	N <sub>U</sub>
1	SOH	Start of Header	A	S <sub>H</sub>
2	STX	Start of Text	B	S <sub>X</sub>
3	ETX	End of Text	C	E <sub>X</sub>
4	EOT	End of Transmission	D	E <sub>T</sub>
5	ENQ	Enquiry	E	E <sub>Q</sub>
6	ACK	Acknowledgement	F	A <sub>K</sub>
7	BEL	Bell	G	B
8	BS	Backspace	H	B <sub>S</sub>
9	HT	Horizontal Tab	I	H <sub>T</sub>
10	LF	Line-feed	J	L <sub>F</sub>
11	VT	Vertical Tab	K	V <sub>T</sub>
12	FF	Form-feed	L	F <sub>F</sub>
13	CR	Carriage-return	M	C <sub>R</sub>
14	SO	Shift Out	N	S <sub>O</sub>
15	SI	Shift In	O	S <sub>I</sub>
16	DLE	Data Link Escape	P	D <sub>L</sub>
17	DC1	Device Control	Q	D <sub>1</sub>
18	DC2	Device Control	R	D <sub>2</sub>
19	DC3	Device Control	S	D <sub>3</sub>
20	DC4	Device Control	T	D <sub>4</sub>
21	NAK	Neg. Acknowledgement	U	N <sub>K</sub>
22	SYN	Synchronous Idle	V	S <sub>Y</sub>
23	ETB	End of Text Block	W	E <sub>B</sub>
24	CAN	Cancel	X	C <sub>N</sub>

Table 11-1. Generating Control Characters with CTRL and ASCII Keys (continued)

Character Code	ASCII Character	Character's Description	Key(s) Pressed with CTRL	Character on CRT
25	EM	End of Media		E <sub>M</sub>
26	SUB	Substitute		S <sub>B</sub>
27	ESC	Escape		E <sub>C</sub>
28	FS	File Separator		F <sub>S</sub>
29	GS	Group Separator		G <sub>S</sub>
30	RS	Record Separator		R <sub>S</sub>
31	US	Unit Separator		U <sub>S</sub>

Pressing the key on the ITF keyboard is an alternative to . The keys listed in the preceding table are **not the only** ways to generate control characters, but are generally the simplest and most easily memorized method. For instance, to generate a line-feed character, press the and the keys simultaneously.

-

Pressing the key with a non-ASCII key is used to generate and store non-ASCII keystrokes within strings and is further discussed in "Outputs to the Keyboard".

On an ITF Keyboard, the display enhancement control codes can be generated by pressing **CTRL**, **Extend char**, and a key from the following table simultaneously.

**Table 11-2. Generating Control Characters with CTRL, Extend char, and ASCII Keys**

Character Code	Character's Description	Key(s) Pressed with CTRL and Extend char	Character on CRT
128	Clear enhancements	<b>1</b>	C <sub>L</sub>
129	Inverse video	<b>2</b>	I <sub>V</sub>
130	Blinking	<b>3</b>	B <sub>G</sub>
131	Inverse blinking	<b>4</b>	I <sub>B</sub>
132	Underline	<b>5</b>	U <sub>L</sub>
133	Underline and Inverse	<b>6</b>	I <sub>V</sub>
134	Underline and Blinking	<b>7</b>	B <sub>G</sub>
135	Underline, Inverse, and Blinking	<b>8</b>	I <sub>B</sub>
136	White	<b>Q</b>	W <sub>H</sub>
137	Red	<b>W</b>	R <sub>D</sub>
138	Yellow	<b>E</b>	Y <sub>E</sub>
139	Green	<b>R</b>	G <sub>R</sub>
140	Cyan	<b>T</b>	C <sub>Y</sub>
141	Blue	<b>Y</b>	B <sub>U</sub>
142	Magenta	<b>U</b>	M <sub>G</sub>
143	Black	<b>I</b>	B <sub>K</sub>



---

## Keyboard Operating Modes

The keyboard has three operating modes which can be changed within a program with the CONTROL statement. This section describes changing these modes from the program.

### The Caps Lock Mode

Pressing the Caps key toggles the keyboard between the "Caps lock on" and "Caps lock off" modes. In the "Caps lock on" mode, pressing any alphabetic key causes an uppercase letter to be displayed on the screen; in the "Caps lock off" mode, these keys generate lowercase letters. This mode can be changed with the CONTROL statement and sensed with the STATUS statement. Writing **any non-zero** numeric value into register 0 (of interface select code 2) sets the caps lock mode **on**; writing a zero into this register sets the mode off.

```
100 STATUS 2;Caps_lock    ! Check mode.
110 !
120 PRINT "Initially, ";
130 IF Caps_lock=1 THEN
140     Mode$="ON"
150 ELSE
160     Mode$="OFF"
170 END IF
180 !
190 PRINT "CAPS LOCK was "&Mode$&CHR$(10)    ! Skip line.
200 BEEP
210 WAIT 1
220 !
230 CONTROL 2;1
240 PRINT "CAPS LOCK now ON"
250 PRINT "Type in a few characters"&CHR$(10)
260 WAIT 4
270 !
280 CONTROL 2;0
290 PRINT "CAPS LOCK now OFF"
300 PRINT
310 BEEP
320 END
```

## The Print All Mode

Pressing the **Prt all** softkey (**f4** in the System menu) toggles the "Print all" mode "on" and "off". The "Print all" mode can also be sensed and changed by reading and writing to STATUS register 1 and CONTROL register 1 (of interface select code 2). Writing a **non-zero** numeric value into this register sets the "Print all" mode **on**; writing a value of zero turns this mode "off". The following statement turns the "Print all" mode off.

```
CONTROL 2,1;0
```

## Disabling Scrolling

If there are results you **do not** want to accidentally scroll off the screen after or while executing a program, keyboard CONTROL register 16 can be used to prevent this from happening. The "scrolling keys" which keyboard register 16 affects are:

- **▲** and **Shift-▲**
- **▼** and **Shift-▼**
- **Prev** and **Shift-Prev**
- **Next** and **Shift-Next**
- **↶** and **Shift-↶**

including implied **▲** and **▼** arrows from knobs and mice, OUTPUT KBD of these keys, and typing-aid softkey definitions which contain these keycodes.

To disable the keys mentioned above, execute the following statement:

```
CONTROL KBD,16;1
```

You can re-enable these keys by writing a 0 (the default state) into this register.

```
CONTROL KBD,16;0
```

The "scrolling keys" are also re-enabled when you:

- power-up your computer.
- press **Shift-Reset**.
- execute either the SCRATCH or SCRATCH A statement.

If you are not sure of the status of the scrolling keys previously mentioned, you can execute the following statement in a program:

```
100 STATUS KBD,16;A
110 END
```

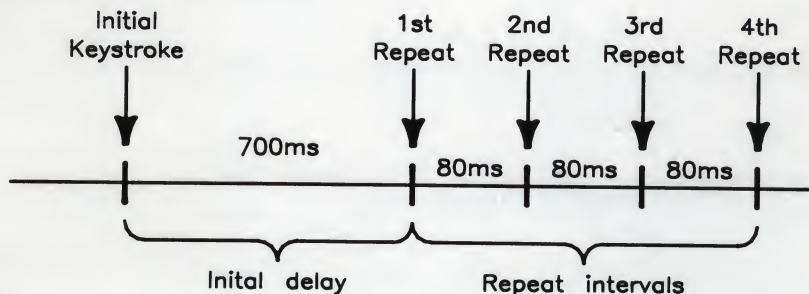
The results returned will be a 1 if the keys are disabled and a 0 if they are enabled.

Note that keyboard register 16 has no effect when you are in the EDIT mode.

Also, programmatic scrolling will still occur as a result of executing TABXY or CONTROL CRT,1;... or printing more lines than fit in the OUTPUT Area.

## Modifying the Repeat and Delay Intervals

The keyboard has an auto-repeat feature which allows you to hold a key down to repeat its function rather than pressing and releasing it repeatedly. Holding a key down will cause it to be repeated every 80 milliseconds for as long as it is held down, resulting in a repeat rate of approximately 12.5 characters per second. However, you may have noticed that the initial delay between the key being pressed and the key being repeated is longer than successive delays between repeats; the initial delay before a key is repeated for the first time is 700 milliseconds (7/10 second). The following plot of a key's **default** repeat function shows these two intervals.



These intervals can be changed from the program, if desired, by writing different values into CONTROL registers 3 and 4 (of interface select code 2). Register 3 contains the parameter that controls the auto-repeat interval, and register 4 contains the parameter that controls the initial delay. The values of these parameters, multiplied by 10, give the respective intervals in milliseconds with the following exception; if register 3 is set to 256, the auto-repeat is disabled.



The following program sets up softkeys 1, 4, 6, 8 to change these parameters. Run the program and experiment with these intervals to optimize them for your own preferences and needs.

---

### Note

Softkey labels (on the keycaps) are **[f1]** through **[f8]** on ITF keyboards. In default mode, the correspondence between key labels (**[f1]**, **[f2]**, etc.) and KEY numbers (in ON KEY and with typing-aid softkeys) is **[f1]**=KEY 1, **[f2]**=KEY 2, etc. You can change this correspondence by writing a 1 into KBD CONTROL register 14; the new correspondence will be **[f1]**=KEY 0, **[f2]**=KEY 1, etc.

---

```
100 ON KEY 1 LABEL "Faster" GOSUB Decr_interval
110 ON KEY 4 LABEL "Slower" GOSUB Incr_interval
120 ON KEY 6 LABEL "Sooner" GOSUB Decr_delay
130 ON KEY 8 LABEL "Later" GOSUB Incr_delay
140 !
150 Interval=80 ! Defaults.
160 Delay=700
170 !
180 DISP "Interval=";Interval;" Delay= ";Delay
190 GOTO 180 ! Loop.
200 !
210 Incr_interval:Interval=Interval+10*(Interval<2560)
220 CONTROL 2,3;Interval/10
230 RETURN
240 !
250 Decr_interval:Interval=Interval-10*(Interval<>10)
260 CONTROL 2,3;Interval/10
270 RETURN
280 !
290 Incr_delay:Delay=Delay+10*(Delay<2560)
300 CONTROL 2,4;Delay/10
310 RETURN
320 !
330 Decr_delay:Delay=Delay-10*(Delay>10)
340 CONTROL 2,4;Delay/10
350 RETURN
360 !
370 END
```



---

## Entering Data from the Keyboard

When the keyboard is specified as the source of data in an ENTER statement, the computer executes the process just as if entering data from any other device. The computer signals to the keyboard that the keyboard is to be the sender of data. The keyboard in turn signals that it is not ready to send data and waits for you to type in and edit the desired data.

The characters you type in appear in the keyboard area of the display, but they are not automatically sent to the computer. As long as you can see the characters, you can edit them before sending them to the computer, **just as during an INPUT statement**. Available characters include all 256 characters that can be generated either with keystrokes or with the **Any char** key (softkey **f7** in the System menu).

Pressing any of the following keys signals the keyboard that the data is to be sent to the computer:

- **Return** key.
- **Enter** key.
- **Step** key (**f1** in the System menu).
- **Continue** key (**f2** in the System menu, and **f2** User 1 and User 2 menus).

The data is then sent byte-serially according to an agreed-upon handshake convention. The computer enters the data in byte-serial fashion and processes it according to the specified variable(s), type of ENTER statement, and image (if it is an ENTER USING statement).

The differences in pressing the keys or softkeys in the above paragraph are as follows. Keep in mind that the ENTER statement is still being executed as long as the “?” appears in the lower right corner of the display.

- Return** Return  
or  
**Enter** Enter  
or  
**Step** f1
- All of the characters displayed in the keyboard area are sent to the computer, followed by carriage-return and line-feed characters. These last two characters **usually** terminate entry into the current item in the ENTER statement. In addition, the **Step** key causes the computer to remain in the single-step mode after the ENTER statement has been completely executed.
- Continue** f2
- All of the characters displayed in the keyboard area are sent to the computer for processing; **no** trailing carriage-return and line-feed characters are sent. The **Continue** key is pressed if more characters are to be entered into the current variable in the destination list of the ENTER statement.

Type in and run the following program. Experiment with how entry into each variable item is terminated by using the different keys (i.e. the **Continue** key versus Return, Enter, or **Step** keys). Pressing the Return, Enter, or **Step** key terminates entry into the current variable, while pressing the **Continue** key allows additional characters to be entered into the current variable.

```
100 DIM String_array$(1:3)[100]
110 ASSIGN @Device_simulate TO 2
120 !
130 ENTER @Device_simulate;String_array$(*)
140 !
150 OUTPUT 1;String_array$(*)
160 !
170 END
```

This use of the keyboard is very powerful when tracing the cause of an error in an ENTER operation. With this tool, you can “debug” or verify any type of ENTER statement, including ENTER statements whose source is intended to be a device on the HP-IB interface. The next section describes this topic.

## Sending the EOI Signal

The EOI signal is implemented on the HP-IB interface. This line ordinarily signals to the computer that the data byte being received is the last byte of the item; thus, it is either an item terminator or a terminating condition for the ENTER statement<sup>1</sup>.

The EOI signal can be simulated from the keyboard when this feature is properly enabled. CONTROL register 12 of interface select code 2 controls this feature; the following example statement shows how to enable this feature.

```
CONTROL 2,12;1
```

To simulate the EOI signal with a character, press the **CTRL** and **E** (or **Shift**-**\*** on the numeric keypad) keys **simultaneously** before the character to be accompanied with EOI is typed. For instance, if the characters "DATA" are to be entered and the EOI is to accompany the last "A", the following key sequence should be pressed before pressing the **Return**, **Enter**, **Step**, or **Continue** key (or softkey).

```
D A T CTRL-E A
```

The same result can be obtained by placing an ENQ character (ASCII control character CHR\$(5), **Eq**) in front of the character to be accompanied by the EOI signal (see the previous section for further details).

---

<sup>1</sup> See the chapter "Entering Data" for a further explanation of the EOI signal's effects during ENTER.



---

## Sending Data to the Keyboard

Characters output to the keyboard are indistinguishable from characters typed in from the keyboard. All characters output to the keyboard, **including control characters**, are displayed in the keyboard area. The following program outputs the BEEP statement to the keyboard. Read on to see how it works.

```
100  OUTPUT 2;"BEEP";  ! No CR/LF
110  !
120  END
```

### Sending Non-ASCII Keystrokes to the Keyboard

The preceding program sent the characters BEEP to the keyboard, but the statement **was not executed**. Pressing the Return or Enter key after the program has ended executes the statement. Modify the program to “press” the Return or Enter key by typing CTRL-Return (or CTRL-Enter) following the BEEP. Sending this special two-character sequence to the keyboard is equivalent to the operator pressing the corresponding key. Thus, **in general**, to store a non-ASCII “keystroke” within a program line, press the CTRL key while simultaneously pressing the desired non-ASCII key.

Since `CHR$(255)` does not generate the same character on most printers as it does on the computer’s display, it is recommended that some explicit means of documenting these character sequences be employed. For instance, string variables can be defined to contain these sequences; then when the program is listed on an external printer, it will be much easier to determine which non-typing keys are being represented. The CTRL key is still used with the non-ASCII key to generate the two-character sequence, but the special character should be changed to a `CHR$(255)`.

```
100  Enter_key$=CHR$(255)&"E"
110  Printall_key$=CHR$(255)&"A"
120  !
130  OUTPUT 2;Printall_key$;  ! Use ";" to suppress CR/LF.
140  OUTPUT 2;"BEEP"&Enter_key$;
150  END
```



---

### Note

Since this type of output can be used to send immediately executed commands (such as **SCRATCH A**), it is important that you use care when outputting commands to the keyboard and when editing statements and commands sent to the keyboard. Undesirable results may occur if the wrong non-ASCII key sequences are output by a program.

---

The table in the next section shows the resultant characters that follow CHR\$(255) in the two-character sequences generated by these keystrokes. The table can be used to look up which non-ASCII key is to be output if the second character is known or vice-versa.

---

## Second Byte of Non-ASCII Key Sequences (String)

Holding the **CTRL** key and pressing a non-ASCII key generates a two-character sequence on the CRT. For example,

**CTRL** - **Clear line**

produces the following characters on the CRT:

**K**

Non-ASCII keypresses can be simulated by outputting these two-byte sequences to the keyboard. For example,

```
OUTPUT KBD;CHR$(255)&"%";
```

produces the same result as shown above. The decimal value of the first byte is 255 (on some computers this is the "inverse-video" **␣**).

The following table can be used to look up the key that corresponds to the second character of the sequence.

Normally on an ITF keyboard, **f1** corresponds to ON KEY 1 ..., **f2** corresponds to ON KEY 2 ..., etc. However, you can use **CONTROL KBD,14;1** to change this relationship so that **f1** corresponds to ON KEY 0..., **f2** corresponds to ON KEY 1, etc.

With 98203 keyboard compatibility (**KBD CMODE ON**), the ITF keyboard softkeys **f1** thru **f4**, the **Menu** and **System** keys, and **f5** thru **f8** correspond to 98203 softkeys **k0** thru **k9**, respectively. See "Porting to Series 300" chapter of *BASIC Programming Techniques* for further information about this mode.

The terms System and User in the **ITF Key** column refer to the softkey menu which is currently active on an ITF keyboard.

Char.	Val.	ITF Key	98203 Key	Closure Key
space	32	1	1	
!	33	Shift-Stop	SHIFT-CLR I/O	Yes
"	34	1	1	
#	35	Shift-Clear line	CLR LN	
\$	36	System f7	ANY CHAR	Yes
%	37	Clear line	CLR→END	Yes
&	38	Select <sup>4</sup>	2	
,	39	Prev	2	Yes
(	40	Shift-Tab	SHIFT-TAB	
)	41	Tab	TAB	
*	42	Insert line	INS LN	Yes
+	43	Insert char	INS CHR	
.	44	Next	2	Yes
-	45	Delete char	DEL CHR	

<sup>1</sup> These characters cannot be generated by pressing the CTRL key and a non-ASCII key. If one of these characters follows CHR\$(255) in an output to the keyboard, an error is reported (**Error 131 Bad non-alphanumeric keycode.**).

<sup>2</sup> Cannot generate this keycode from this keyboard. If this character is OUTPUT to the keyboard, an error is **not** reported. Instead, the system will perform as much of the indicated action as possible.

<sup>4</sup> These keys have no system meaning, and will BEEP if not trapped by ON KBD.

Char.	Val.	ITF Key	98203 Key	Closure Key
.	46	2	2	
/	47	Delete line	DEL LN	Yes
0	48	User 3 f8	k0	Yes
1	49	User 1 f1	k1	Yes
2	50	User 1 f2	k2	Yes
3	51	User 1 f3	k3	Yes
4	52	User 1 f4	k4	Yes
5	53	User 1 f5	k5	Yes
6	54	User 1 f6	k6	Yes
7	55	User 1 f7	k7	Yes
8	56	User 1 f8	k8	Yes
9	57	User 2 f1	k9	Yes
:	58	System Shift-f6 <sup>4</sup>	2	
;	59	System Shift-f7 <sup>4</sup>	2	
<	60	←	←	
=	61	Result <sup>3</sup>	RESULT	
>	62	→	→	
?	63	Recall <sup>3 6</sup>	RECALL	
@	64	Shift-Recall <sup>3 7</sup>	SHIFT-RECALL	
A	65	System f4	PRT ALL	Yes
B	66	Back space	BACK SPACE	
C	67	System f2	CONTINUE	

<sup>2</sup> Cannot generate this keycode from this keyboard. If this character is OUTPUT to the keyboard, an error is **not** reported. Instead, the system will perform as much of the indicated action as possible.

<sup>3</sup> This ITF key is located in the System Control Key Group just above the Numeric Keypad Group. Note that these keys have no labels on their keycaps; however, they do have labels on the BASIC keyboard overlay for the ITF keyboard. For information on the BASIC keyboard overlay for the ITF keyboard, read the manual entitled *Using the BASIC/UX System*.

<sup>4</sup> These keys have no system meaning, and will BEEP if not trapped by ON KBD.

<sup>6</sup> also System f8

<sup>7</sup> also System Shift-f8

## 11-18 The Keyboard Interface



Char.	Val.	ITF Key	98203 Key	Closure Key
D	68	2	EDIT	
E	69	Enter <sup>8</sup>	ENTER	Yes
F	70	System f6	DISPLAY FCTNS	Yes
G	71	Shift-▶	SHIFT-➡	
H	72	Shift-◀	SHIFT-⬅	
I	73	Break	CLR I/O	
J	74	(Katakana) <sup>2</sup>	(Katakana) <sup>2</sup>	
K	75	Clear display	CLR SCR	Yes
L	76	Graphics <sup>3</sup>	GRAPHICS	Yes
M	77	Alpha <sup>3</sup>	ALPHA	Yes
N	78	Dump Graph <sup>3</sup>	DUMP GRAPHICS	Yes
O	79	Dump Alpha <sup>3 9</sup>	DUMP ALPHA	Yes
P	80	Stop	PAUSE	Yes
Q	81	1	1	
R	82	System f3	RUN	Yes
S	83	System f1	STEP	Yes
T	84	Shift-▼	SHIFT-⬇	Yes
U	85	Caps	CAPS LOCK	Yes
V	86	▼	⬇	Yes
W	87	Shift-▲	SHIFT-⬆	Yes
X	88	2	EXECUTE	Yes

<sup>1</sup> These characters cannot be generated by pressing the CTRL key and a non-ASCII key. If one of these characters follows CHR\$(255) in an output to the keyboard, an error is reported (**Error 131 Bad non-alphanumeric keycode.**).

<sup>2</sup> Cannot generate this keycode from this keyboard. If this character is OUTPUT to the keyboard, an error is **not** reported. Instead, the system will perform as much of the indicated action as possible.

<sup>3</sup> This ITF key is located in the System Control Key Group just above the Numeric Keypad Group. Note that these keys have no labels on their keycaps; however, they do have labels on the BASIC keyboard overlay for the ITF keyboard. For information on the BASIC keyboard overlay for the ITF keyboard, read the manual entitled *Using the BASIC/UX System*.

<sup>8</sup> Or **Return**

<sup>9</sup> Also **Print**

Char.	Val.	ITF Key	98203 Key	Closure Key
W	87	Shift-▲	SHIFT-↑	Yes
X	88	2	EXECUTE	Yes
Y	89	(Roman) <sup>2</sup>	(Roman) <sup>2</sup>	Yes
Z	90	1	1	
[	91	System f5	CLR TAB	
\	92	▼	2	Yes
]	93	System Shift-f5	SET TAB	
^	94	▲	↑	Yes
-	95	System Shift-▼	2	Yes
'	96	1	1	
a	97	User 2 f2	SHIFT-k0	Yes
b	98	User 2 f3	SHIFT-k1	Yes
c	99	User 2 f4	SHIFT-k2	Yes
d	100	User 2 f5	SHIFT-k3	Yes
e	101	User 2 f6	SHIFT-k4	Yes
f	102	User 2 f7	SHIFT-k5	Yes
g	103	User 2 f8	SHIFT-k6	Yes
h	104	User 3 f1	SHIFT-k7	Yes
i	105	User 3 f2	SHIFT-k8	Yes
j	106	User 3 f3	SHIFT-k9	Yes
k	107	User 3 f4	2	Yes
l	108	User 3 f5	2	Yes
m	109	User 3 f6	2	Yes

<sup>1</sup> These characters cannot be generated by pressing the CTRL key and a non-ASCII key. If one of these characters follows CHR\$(255) in an output to the keyboard, an error is reported (**Error 131 Bad non-alphanumeric keycode.**).

<sup>2</sup> Cannot generate this keycode from this keyboard. If this character is OUTPUT to the keyboard, an error is not reported. Instead, the system will perform as much of the indicated action as possible.

Char.	Val.	ITF Key	98203 Key	Closure Key
n	110	User 3 <b>f7</b>	2	Yes
o	111	System <b>Shift-f1</b> <sup>4</sup>	2	
p	112	System <b>Shift-f2</b> <sup>4</sup>	2	
q	113	System <b>Shift-f3</b> <sup>4</sup>	2	
r	114	System <b>Shift-f4</b> <sup>4</sup>	2	
s	115	User <b>Shift-f1</b> <sup>4 5</sup>	2	
t	116	User <b>Shift-f2</b> <sup>4 5</sup>	2	
u	117	User <b>Shift-f3</b> <sup>4 5</sup>	2	
v	118	User <b>Shift-f4</b> <sup>4</sup>	2	
w	119	User <b>Shift-f5</b> <sup>4</sup>	2	
x	120	User <b>Shift-f6</b> <sup>4</sup>	2	
y	121	User <b>Shift-f7</b> <sup>4</sup>	2	
z	122	User <b>Shift-f8</b> <sup>4</sup>	2	
{	123	<b>System</b>	2	Yes
	124	<b>Menu</b>	2	Yes
}	125	<b>User</b>	2	Yes
~	126	<b>Shift-Menu</b>	2	Yes
	127	1	1	

<sup>1</sup> These characters cannot be generated by pressing the CTRL key and a non-ASCII key. If one of these characters follows CHR\$(255) in an output to the keyboard, an error is reported (**Error 131 Bad non-alphanumeric keycode.**).

<sup>2</sup> Cannot generate this keycode from this keyboard. If this character is OUTPUT to the keyboard, an error is **not** reported. Instead, the system will perform as much of the indicated action as possible.

<sup>4</sup> These keys have no system meaning, and will BEEP if not trapped by ON KBD.

<sup>5</sup> These keys are also generated by the HP 46060A/B and HP 46095A (HP Mouse devices) buttons unless GRAPHICS INPUT IS is using them.



## Closure Keys

Several of the non-ASCII keys are known as "closure keys"<sup>1</sup>. Closure keys are so named because they close (block) further keyboard input until processed. **The computer can only process two closure keys between program lines during a running program.** If more than two appear in the data output to the keyboard, the extra keys will be deferred until the next end-of-line is encountered and two more closure keys can be processed.

As an example, the following program sends four closure keys to the keyboard with a single OUTPUT statement. Only the first two closure keys are processed **after** this OUTPUT statement (but **before** DISP "Next BASIC line" is executed). The third and fourth closure keys are processed after DISP "Next BASIC line" is executed (but before DISP "2nd BASIC line" is executed). This accounts for the following display after running the program, since the "Printall" command was not executed until after DISP "Next BASIC line" was executed.

```
100  !    Define non-ASCII keys.
110  En$=CHR$(255)&"E"    ! ENTER or Return key.
120  Up$=CHR$(255)&"^"    ! Up arrow key.
130  Prt$=CHR$(255)&"A"   ! PRT ALL key or softkey.
140  !
150  CONTROL 2,1;0    ! Turn PRINTALL off.
160  CONTROL 1,1;1    ! Begin on top screen line.
170  OUTPUT 1;"Line 1"
180  OUTPUT 1;"Line 2"
190  OUTPUT 1;"Line 3"
200  WAIT 1
210  !
220  !    Now send statement with 4 closure keys.
230  OUTPUT 2;"DISP ""Hello""";En$;Up$;Up$;Prt$;
240  DISP "Next BASIC line" ! PRT ALL still off.
250  DISP "2nd BASIC line"  ! Now PRT ALL is on.
260  !
270  END
```

---

<sup>1</sup> See the table on the preceding pages to determine which keys are "closure keys".



## Display After Running Program

Line 3  
2nd BASIC line

2nd BASIC line

Printall on

In addition, if the last character sent to the keyboard is a CHR\$(255), the next character typed in by the user will give unexpected results. Again, it is important to exercise care when using this feature.

---

## Softkeys

The keys on the upper-left portion of the keyboard are called “softkeys.” These keys can be defined by BASIC programs to initiate program branches. In addition, these keys can be defined as typing-aid keys, which produce keystrokes just as if you had typed them in yourself.

Brief examples of using the softkeys have already been presented in the “Interface Events” chapter, and in the section found earlier this chapter entitled “Modifying the Repeat and Delay Intervals”. Typing-aid softkeys are discussed in the chapter “Introduction to the System” found in the *Using the BASIC/UX System* manual. Softkeys are also briefly described in the “Program Structure and Flow” and “Communicating with the Operator” chapters of the *BASIC Programming Techniques* manual.

## Sensing Knob Rotation

Your computer system may or may not have a knob (built-in, or HP 46083) or a mouse (HP 46060). In any event, the programs below are illustrative of how knob and mouse movements can be trapped in a program. It is assumed that you will use the techniques and apply them to your programming situation.

The “event” of the knob (rotary pulse generator) being rotated can be sensed by a program. The branch location, interval at which the computer interrogates the knob for the occurrence of rotation, and branch priority are set up with a statement such as the following:

```
ON KNOB Interval,Priority CALL Knob_turned
```

In addition to the program being able to sense rotations of the knob, it can also determine how many pulses the knob has produced and whether or not either or both of the **CTRL** or **Shift** keys are being pressed<sup>1</sup>. This ability to “qualify” the use of the knob allows it to be used for up to four different purposes. The following program shows how to set up the branch, how to determine the number of pulses, and how to determine the direction of rotation.

```
100  ON KNOB .25 GOSUB Knob  ! Check knob every 1/4 sec.
110  !
120  FOR Iteration=1 TO 400
130    WAIT .2
140    DISP Iteration
150  NEXT Iteration
160  !
170  STOP
180  !
190 Knob:  STATUS 2,10;Key_with_knob
200        PRINT KNOBX;" pulses ";KNOBY;" pulses ";
201        DISP TAB(40),"Status = ";Key_with_knob
210        IF Key_with_knob=0 THEN
220          PRINT
230        ELSE
240          IF Key_with_knob=1 THEN PRINT "with SHIFT"
250          IF Key_with_knob=2 THEN PRINT "with CTRL"
260          IF Key_with_knob=3 THEN PRINT "with SHIFT and CTRL"
270        END IF
280        RETURN
290  END
```

<sup>1</sup> HIL devices do not set the “CTRL” bit, although they do set the “SHIFT” bit (if the last record processed was “y-axis” data). Consequently, you should not depend on the value of keyboard status register 10.

If any pulses have occurred since the last branch, the specified branch will be initiated.

One full rotation of the knob produces 120 pulses. The service routine calls the KNOBX and KNOBY functions to determine how many pulses (only **net** rotation) have been generated **since the last call** to this function. If the number is positive, a net clockwise rotation has occurred; a negative number signifies that a net counterclockwise rotation has occurred. Since the pulse counter (on built-in knobs) can only sense +128 to -127 pulses **during the specified interval**<sup>1</sup>, the interval parameter should be chosen small enough to interrogate the knob before the pulse counter reaches one of these values. **Experiment** with this parameter to adjust it for your particular application.

The next program illustrates the use of an ON KNOB with a mouse (HP 46060). Note changes in iteration as you move the mouse.

```
10    COM /Knob/ Kx,Ky
20    Kx=0
30    Ky=0
40    ON KNOB 1 CALL Knob
50    PRINT TABXY(1,1);" "
60    FOR I=1 TO 1.E+6
70        DISP I
80        PRINT TABXY(1,2);Kx;Ky;"          "
90    NEXT I
100   END
110   SUB Knob
120       COM /Knob/ Kx,Ky
130       INTEGER Knx,Kny
140       Knx=KNOBX
150       Kny=KNOBY
160       Kx=Kx+Knx
170       Ky=Ky+Kny
180       PRINT TABXY(1,5);Knx;Kny;"          "
190   SUBEND
```

You can also trap mouse keys with ON KBD and KBD\$ function (see the subsequent section for details on using these keywords). These keys produce the same codes as the Shift-f1, Shift-f2, etc. keys on ITF keyboards (while in any User menu).

---

<sup>1</sup> HIL devices can count from 32 767 to -32 768 pulses during the interval.



---

## Enhanced Keyboard Control

Normally, the BASIC operating system handles all keyboard inputs. Several BASIC statements allow programs to handle inputs from the keyboard; examples are the INPUT, LINPUT, ENTER, ON KEY, and ON KNOB statements. Additional keyboard statements provide BASIC programs with a means of intercepting both ASCII and non-ASCII keystrokes for processing by the program. The statements are:

- ON KBD                sets up and enables keystrokes to be trapped.
- ON KBD ,ALL        includes **Pause**, **Stop**, **Clr I/O**, **System**, **User**, **Menu**, **Shift-Menu** and softkeys. See the key tables in the section of this chapter entitled "Second Byte of Non-ASCII Key Sequences (String)" for appropriate ITF key labels.
- KBD\$                returns keystrokes trapped in the buffer.
- OFF KBD            resumes normal keystroke processing.

ON KBD allows terminal emulation, keyboard masking, and special data inputs. Each keystroke produces unique code(s) that allow the program to differentiate between different keys being pressed. The program can also determine whether the **Shift** or **CTRL** keys are being pressed with most keys, but these keystrokes cannot be detected by themselves. Also, the **Reset** key cannot be trapped by ON KBD.

### Trapping Keystrokes

The ON KBD statement sets up a branch that is initiated when the keyboard buffer becomes "non-empty". The service routine may then interrogate the buffer as desired, processing the keystrokes as determined by the program. The keyboard buffer contains up to 256 characters. Calling the KBD\$ function does two things: it returns all keystrokes trapped since the last time the buffer was read, and it then clears the keyboard buffer.

The following program uses ON KBD, KBD\$, and OFF KBD to trap and process keystrokes, rather than allowing the operating system to do the same. The program defines each keystroke to print a complete word.

```

100  OPTION BASE 1
110  DIM String$(26) [6]
120  READ String$(*)
130  !
140  DATA A,BROWN,CAT,DOG,EXIT,FOX,GOT
150  DATA HI,IN,JUMPS,KICKED,LAZY,MY
160  DATA NO,OVER,PUSHED,QUICK,RED,SMART
170  DATA THE,UNDER,VERY,WHERE,XRAY,YES,ZOO
180  !
190  PRINTER IS 1
200  PRINT "Many ASCII keys have been"
210  PRINT "defined to produce words."
220  PRINT
230  PRINT "Press the following keys."
240  PRINT "T Q B F J O T L D ."
250  !
260  ON KBD GOSUB Process_keys
270  !
280  LOOP
290  EXIT IF Word$="EXIT"
300  END LOOP
310  !
320  STOP
330  !
340  Process_keys:Key$=KBD$      ! Read buffer.
350  !
360  REPEAT ! Process ALL keys trapped.
370    Key_code=NUM(Key$[1;1]) ! Calculate code.
380    !
390    SELECT Key_code          ! Choose response.
400    !
410    CASE 65 TO 90            ! CASE "A" TO "Z".
420      Word$=String$(Key_code-64)
430      Key$=Key$[2]          ! Remove processed key.
440      !

```

```

450 CASE 97 TO 122           ! CASE "a" TO "z".
460   Word$=String$(Key_code-96)
470   Key$=Key$[2]         ! Remove processed key.
480   !
490 CASE 255               ! CASE non-ASCII key.
500   IF Key$[2;1]<>CHR$(255) THEN
510     Word$=Key$[1,2]     ! Non-ASCII key alone,
520     Key$=Key$[3]       ! so take 2 codes.
530   ELSE
540     Word$=Key$[1,3]     ! Non-ASCII w/ CTRL,
550     Key$=Key$[4]       ! so take 3 codes.
560   END IF
570 CASE ELSE              ! CASE all others.
580   Word$=""
590   Key$=Key$[2]         ! Remove processed key.
600   !
610 END SELECT
620 !
630                       ! Execute response.
640 Defined=LEN(Word$)<>0
650 IF Defined THEN
660   PRINT Word$;" ";
670   DISP
680 ELSE
690   BEEP 100,.05
700   DISP "Key undefined."
710 END IF
720 !
730 UNTIL LEN(Key$)=0      ! Until ALL keys processed.
740 !
750 RETURN
760 !
770 Quit:END

```

Notice that all non-ASCII keys produce two-character sequences: CHR\$(255) followed by an ASCII character. Pressing the CTRL key with non-ASCII keys produce three-character sequences: another CHR\$(255) character preceding the two-character sequence produced by pressing the non-ASCII key by itself. See the tables in the section entitled "Second Byte of Non-ASCII Key Sequences (String)" for a listing of the sequences produced by non-ASCII keys.



BASIC programs can output ASCII keystrokes to the keyboard, via **OUTPUT 2**, without initiating an **ON KBD** branch; however, outputting non-ASCII “closure” keys followed by other keys will initiate the **ON KBD** branch. For example, executing the following statement (in a program line):

```
OUTPUT 2;"32*2";CHR$(255);"E";"KBD";
```

causes the characters **KBD** which follow the closure key to be placed in the **KBD\$** buffer, which also initiates the **ON KBD** branch. The Return keycode which was sent to the keyboard executes the numeric expression **32\*2** before the branch is initiated. **OUTPUT** to the keyboard while **ON KBD** is in effect should contain at most one closure key, and that key should be at the end, in order to avoid this “recirculation” of closure keys.

**ON KBD** branching is disabled by **DISABLED**, deactivated by **OFF KBD**, and temporarily deactivated when the program is executing **LINPUT**, **INPUT**, or **ENTER KBD** statements. Note that the keyboard input line can be read without deactivating **ON KBD** by using the **SYSTEM\$**(“**KBD LINE**”) function.

## Mouse Keys

You can also trap mouse keys with this technique. The keys produce **CHR\$(255)** followed by “s”, “t”, and so forth.

## Softkeys and Knob Rotation

When **ON KNOB** is not in effect, knob rotation is also trapped by **ON KBD**. Rotation of the knob will produce “cursor” keystrokes. A clockwise rotation of the knob produces **CHR\$(255)** followed by “>”, while a counter-clockwise rotation produces **CHR\$(255)** followed by “<”. When using the HP 46083 Rotary Control Knob, pressing the Shift key and rotating the Knob clockwise produces **CHR\$(255)** followed by “^”, and rotating the Knob counter-clockwise produces **CHR\$(255)** followed by “V”. These same results can be produced when using the HP 46060A Mouse; however, the results are dependent on the “toggle” state for the Rotary Control Knob and “horizontal” and “vertical” movements for the HP Mouse.

**ON KBD ,ALL** allows softkey trapping (“overrides” **ON KEY**) but does not change the softkey labels.



## Disabling Interactive Keyboard

Another group of statements is used to disable the interactive keyboard functions:

SUSPEND INTERACTIVE	ignores the <b>Pause</b> , <b>Stop</b> , <b>Step</b> , and <b>Clr I/O</b> keys (see the table in the section entitled "Second Byte of Non-ASCII Key Sequences (String)" for equivalent ITF keys) and disables live keyboard execution.
SUSPEND INTERACTIVE, RESET	ignores <b>Reset</b> (see the table in the section entitled "Second Byte of Non-ASCII Key Sequences (String)" for equivalent ITF key) too.
RESUME INTERACTIVE	returns to normal operation.

SUSPEND INTERACTIVE can be used to prevent interruption of programs which gather data or which control other systems.

Special care should be taken when using SUSPEND INTERACTIVE, RESET. If an "infinite loop" is executed while interactive keyboard functions are disabled, only the power switch will stop execution of the program.

```
110  ! This program cannot be stopped by
120  ! Pause, Stop, or Reset keys
130  ! before its normal completion.
140  !
150  !
160  SUSPEND INTERACTIVE, RESET ! Ignore keyboard.
170  !
180  PRINT "COUNTDOWN IS "
190  PRINT
200  I=10                      ! Initial value.
210  REPEAT
220    PRINT " T minus ";I    ! Print count.
230    I=I-1                  ! Decrement count.
240    WAIT 1                  ! Wait one second.
250  UNTIL I<0
260  !
270  PRINT
280  BEEP 100,1
290  PRINT "Done"
300  RESUME INTERACTIVE      ! Return to normal.
310  !
320  END
```

---

## Locking Out the Keyboard

There are certain times during program execution when it is expedient to prevent the operator from using the keyboard, such as during a critical experiment which cannot be disturbed. Then the knob and groups of keyboard keys can be enabled and disabled separately.

Setting bit 0 of register 7 (of interface select code 2) **disables** all keys (excluding the **Reset** key) and the knob. The following program first sets up the KNOB and KEY events to initiate program branches. It is assumed that the keyboard is already enabled; if you are not sure, press the **Reset** key. When the program is run, the keyboard and knob remain enabled for about five seconds, after which they are disabled. The program then displays the time of day indefinitely; the only way to stop the program is to press the **Reset** key.

```
100  ON KEY 1 LABEL "SFK 1" GOSUB Key1
110  ON KNOB .2 GOSUB Knob
120  !
130  PRINT "You've got 5 seconds.  GO!  "
140  FOR Iteration=1 TO 20
150    WAIT .25
160  NEXT Iteration
170  !
180  Reset_disable=0    ! Reset key remains ENABLED.
190  Ky_knb_disable=1  ! DISABLE reset of kbd.
200  CONTROL 2,7;2*Reset_disable+Ky_knb_disable
210  PRINT "Time's up!"
220  BEEP
230  !
240 Loop: DISP TIME$(TIMEDATE)
250    GOTO Loop
260    !
270    !
280 Key1: PRINT "Special function key 1 pressed."
290    RETURN
300    !
310 Knob: PRINT "Knob rotation sensed."
320    RETURN
330  END
```

If the value of the variable `Reset_disable` is set to 1 in the preceding program, the only way to **stop** the program is to turn off power to the computer, losing the program and all data currently in computer memory.

---

#### Note

Use care when locking out **both** the `Reset` key and the keyboard keys. If both are locked out, the **only** way to prematurely stop the program is to turn the computer off.

---

### Special Considerations

Disabling keyboard interrupts by locking out the keyboard will also block the use of other HP-HIL devices. For example, if an HP-HIL Graphics Tablet is the current graphics input device and keyboard interrupts are disabled, executing a `DIGITIZE` statement will cause the system to hang, waiting for a response it cannot receive. Attempting to execute an `HIL SEND` statement while keyboard interrupts are disabled will cause an error to occur.

---

## Keyboard Status and Control Registers

<b>STATUS Register 0</b>	CAPS lock flag
<b>CONTROL Register 0</b>	Set CAPS lock if non-0
<b>STATUS Register 1</b>	PRINTALL flag
<b>CONTROL Register 1</b>	Set PRINTALL if non-0
<b>STATUS Register 2</b>	Function key menu.
<b>CONTROL Register 2</b>	Function key menu: 0 = System menu (or SYSTEM KEYS statement) 1-3 = User menu 1 thru 3 (or USER <i>n</i> KEYS statement along with the appropriate menu number)
<b>STATUS Register 3</b>	Undefined
<b>CONTROL Register 3</b>	Set auto-repeat interval. If 1 thru 255, repeat interval in milliseconds is 10 times this value. 256 = turn off auto-repeat. (Default at entry to BASIC/UX or SCRATCH A is the value that was in effect before entry to BASIC/UX.)
<b>STATUS Register 4</b>	Undefined
<b>CONTROL Register 4</b>	Set delay before auto-repeat. If 1 thru 256, delay in milliseconds is 10 times this value. (Default at entry to BASIC/UX or SCRATCH A is the value that was in effect before entry to BASIC/UX.)
<b>STATUS Register 5</b>	KBD\$ buffer overflow register. 1 = overflow Register is reset when read.
<b>CONTROL Register 5</b>	Undefined
<b>STATUS Register 6</b>	Typing aid expansion overflow register. 1 = overflow. Register is reset when read.
<b>CONTROL Register 6</b>	Undefined



**STATUS Register 7      Interrupt Status****Most Significant Bit****Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	INITIAL- IZE Timeout Interrupt Disabled	Reserved For Future Use	Reserved For Future Use	Reset Key Interrupt Disabled	Keyboard and Knob Interrupt Disabled
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**CONTROL Register 7      Interrupt Disable Mask****Most Significant Bit****Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used			INITIAL- IZE Timeout	Reserved For Future Use	Reserved For Future Use	Reset Key	Keyboard and Knob
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**STATUS Register 8****Keyboard Language Jumper**

0-US ASCII	7-United Kingdom	13-Swiss German
1-French	8-Canadian French	14-Latin(Spanish)
2-German	9-Swiss French	15-Danish
3-Swedish	10-Italian	16-Finnish
4-Spanish	11-Belgian	17-Norwegian
5-Katakana	12-Dutch	18-Swiss French*
6-Canadian English		19-Swiss German*

See also SYSTEM\$(“KEYBOARD LANGUAGE”) which requires the LEX binary. Note that the STATUS statement when used with this register does not require the LEX binary.

**CONTROL Register 8**    Undefined

**STATUS Register 9**    Keyboard Type

**Most Significant Bit**

**Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Internal Use		1=HIL Keyboard Interface	1=No Keyboard	1=n-Key Rollover	0	0	0
		0=non- HIL	0=Key- board Present	0=2 or less rollover			
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Bits 5, 1, and 0 of STATUS Register 9 and the following table can be used to determine the Keyboard Type.

Bit 5	Bit 1	Bit 0	Keyboard Type
0	0	0	HP 98203B or built-in (unsupported)
0	0	1	HP 98203A (unsupported)
1	0	0	ITF (such as the HP 46020A and 46021A)
1	1	0	HP 98203C (unsupported)

**CONTROL Register 9** Undefined

**STATUS Register 10** Status at Last Knob Interrupt

Most Significant Bit

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	CTRL Key Pressed	SHIFT Key Pressed
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Note that bit 1 is *always* 0 for keyboards and all HP-HIL mice and knobs (e.g. HP 46083A Rotary Control Knob and HP 46085 Control Dials).

**CONTROL Register 10** Undefined

**STATUS Register 11** 0=horizontal-pulse mode; 1=all-pulse mode.

**CONTROL Register 11** Set knob pulse mode. (This CONTROL register is *not* supported with BASIC/UX, because the KNB2\_0 binary is unsupported on BASIC/UX.)

**STATUS Register 12** "Pseudo-EOI for CTRL-E" flag

**CONTROL Register 12** Enable pseudo-EOI for CTRL-E if non-0

**STATUS Register 13** Katakana flag

**CONTROL Register 13** Set Katakana if non-0

- STATUS Register 14**      Numbering of softkeys on ITF keyboard:  
0  $\Rightarrow$   is key number 1 (default);  
1  $\Rightarrow$   is key number 0;
- CONTROL Register 14**      Softkey numbering on ITF keyboard (see above register description).
- STATUS Register 15**      Currently in 98203 keyboard compatibility mode:  
0  $\rightarrow$  OFF (default)  
1  $\rightarrow$  ON
- CONTROL Register 15**      Turns "98203 keyboard compatibility mode" on ( $\neq 0$ ) and off ( $= 0$ ). (See the chapter "Porting to Series 300" in the *Programming Techniques* manual for further information about using this mode.) Note that instead of using the CONTROL register 15 statement you can use the KBD CMODE statement to turn the "98203 keyboard compatibility mode" ON and OFF.
- STATUS Register 16**      Returns the enabled/disabled status of the up and down arrow keys, , , and  (both shifted and un-shifted for all of these keys). If the status value is 1 it means these keys are deactivated. Note that the default value is 0.
- CONTROL Register 16**      Allows you to disable or re-enable the display scrolling keys mentioned for STATUS Register 16. This prevents accidental scrolling of the display screen. Executing a 1 with the CONTROL statement deactivates the print scrolling keys and a 0 activates them.
- STATUS Register 17**      Automatic menu switching:  
1  $\Rightarrow$  enabled (default)  
0  $\Rightarrow$  disabled
- CONTROL Register 17**      Automatic menu switching:  
 $\langle \rangle 0 \Rightarrow$  enable  
0  $\Rightarrow$  disable
- This register controls whether a system with an ITF keyboard will switch to (from) the User 2 Menu automatically on entering (leaving) EDIT mode.



# Table of Contents

---

## Chapter 12: The HP-IB Interface

Introduction .....	12-1
Initial Installation and Verification .....	12-2
Communicating with Devices .....	12-3
HP-IB Device Selectors .....	12-3
Moving Data Through the HP-IB .....	12-4
Using an Interface in the HP-UX Environment .....	12-6
General Structure of the HP-IB .....	12-8
Addressing Multiple Listeners .....	12-11
Secondary Addressing .....	12-11
General Bus Management .....	12-12
Remote Control of Devices .....	12-13
Locking Out Local Control .....	12-14
Enabling Local Control .....	12-14
Triggering HP-IB Devices .....	12-15
Clearing HP-IB Devices .....	12-15
Aborting Bus Activity .....	12-16
HP-IB Service Requests .....	12-16
Polling HP-IB Devices .....	12-19
Advanced Bus Management .....	12-21
The Message Concept .....	12-21
Types of Bus Messages .....	12-22
Explicit Bus Messages .....	12-27
HP-IB Message Mnemonics .....	12-30
The Computer As a Non-Active Controller .....	12-32
Determining Controller Status and Address .....	12-32
Changing the Controller's Address .....	12-34
Passing Control .....	12-34
Interrupts While Non-Active Controller .....	12-35
Addressing a Non-Active Controller .....	12-39
Requesting Service .....	12-41
Responding to Parallel Polls .....	12-42
Responding to Serial Polls .....	12-44
Interface-State Information .....	12-45
Servicing Interrupts that Require Data Transfers .....	12-46
HP-IB Control Lines .....	12-49

Handshake Lines .....	12-50
The Attention Line (ATN) .....	12-50
The Interface Clear Line (IFC) .....	12-51
The Remote Enable Line (REN) .....	12-51
The End or Identify Line (EOI) .....	12-51
The Service Request Line (SRQ) .....	12-52
Determining Bus-Line States .....	12-53
Summary of HP-IB STATUS and CONTROL Registers .....	12-54
HP-IB Status and Control Registers (cont.) .....	12-55
HP-IB Status and Control Registers (cont.) .....	12-56
HP-IB Status and Control Registers (cont.) .....	12-57
HP-IB Status and Control Registers (cont.) .....	12-58
HP-IB Status and Control Registers (cont.) .....	12-59
Summary of HP-IB READIO and WRITEIO Registers .....	12-60
READIO Registers .....	12-60
HP-IB WRITEIO Registers .....	12-67
Summary of Bus Sequences .....	12-73
ABORT .....	12-73
CLEAR .....	12-74
LOCAL .....	12-74
LOCAL LOCKOUT .....	12-74
PASS CONTROL .....	12-75
PPOLL .....	12-75
PPOLL CONFIGURE .....	12-75
PPOLL UNCONFIGURE .....	12-76
REMOTE .....	12-76
SPOLL .....	12-77
TRIGGER .....	12-77

# The HP-IB Interface

# 12

## Introduction

This chapter describes the techniques necessary for programming the HP-IB interface. Many of the elementary concepts have been discussed in previous chapters; this chapter describes the specific details of how this interface works and how it is used to communicate with and control systems consisting of various HP-IB devices.

The HP-IB (Hewlett-Packard Interface Bus), commonly called the “bus”, provides compatibility between the computer and external devices conforming to the IEEE 488-1978 standard. Electrical, mechanical, and timing compatibility requirements are all satisfied by this interface.

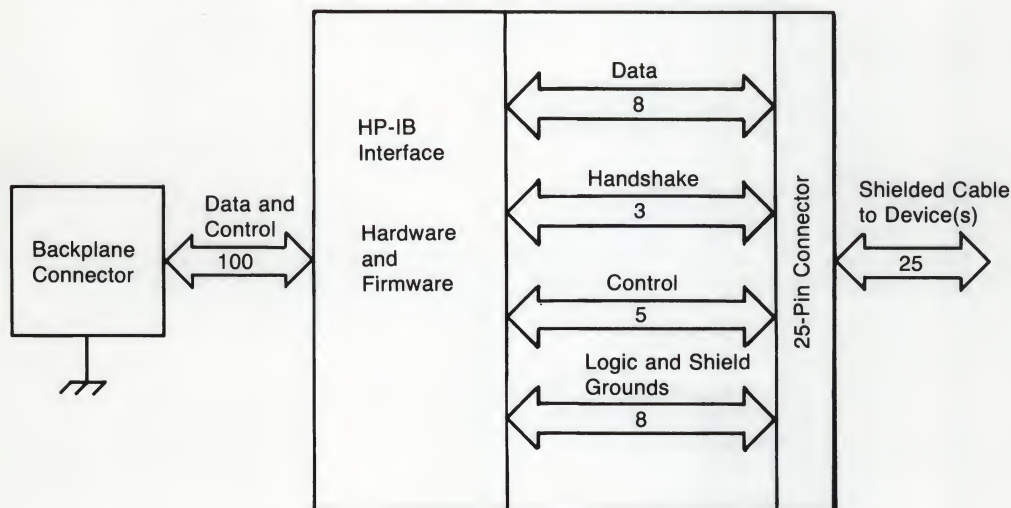


Figure 12-1. HP-IB Interface Block Diagram

The HP-IB Interface is both easy to use and allows great flexibility in communicating data and control information between the computer and external devices. It is one of the easiest methods to connect more than one device to the same interface.



---

## Initial Installation and Verification

Refer to the HP-IB Installation Note for information about setting the switches and installing an external HP-IB interface. Once the interface has been properly installed, you can verify that the switch settings are what you intended by running the following program. The defaults of the internal HP-IB interface can also be checked with the program. The results are displayed on the CRT.

```
100  PRINTER IS CRT
110  PRINT CHR$(12) ! Clear screen w/ FF.
120  !
130  Ask: INPUT "Enter HP-IB interface select code",Isc
140  IF Isc<7 OR Isc>30 THEN GOTO Ask
150  !
160  STATUS Isc;Card_id
170  IF Card_id<>1 THEN
180      PRINT "Interface at select code";Isc;
190      PRINT "is not an HP-IB"
200      PRINT
210      STOP
220  END IF
230  !
240  PRINT "HP-IB interface present"
250  PRINT " at select code";Isc
260  PRINT
270  !
280  STATUS Isc,1;Intr_dma
290  Level=3+(BINAND(32+16,Intr_dma) DIV 16)
300  PRINT "Hardware interrupt level =";Level
310  !
320  STATUS Isc,3;Addr_ctrlr
330  Address=Addr_ctrlr MOD 32
340  PRINT "Primary address =";Address
350  !
360  Sys_ctrl=BIT(Addr_ctrlr,7)
370  IF Sys_ctrl THEN
380      PRINT "System Controller"
390  ELSE
400      PRINT "Non-system Controller"
410  END IF
420  !
430  END
```

The hardware interrupt level is described in Chapter 7. Hardware interrupt level is set to 3 on built-in HP-IB interface, but can range from 3 to 6 on optional interfaces. Primary address is further described in "HP-IB Device Selectors" in the next section.



The term "System Controller" is also further described later in this chapter in "General Structure of the HP-IB". The internal HP-IB has a jumper or switch that is set at the factory to make it a system controller. To find out the location of this jumper or switch, refer to the documentation that comes with your computer. Note that the location varies with different Models of computers. External HP-IB interfaces have a switch that controls this interface state.

---

## Communicating with Devices

This section describes programming techniques used to output data to and enter data from HP-IB devices. General bus operation is also briefly described in this chapter. Later chapters will describe: further details of specific bus commands, handling interrupts, and advanced programming techniques.

### HP-IB Device Selectors

Since the HP-IB allows the interconnection of several devices, each device must have a means of being uniquely accessed. Specifying just the interface select code of the HP-IB interface through which a device is connected to the computer is not sufficient to uniquely identify a specific device on the bus.

Each device "on the bus" has an **primary address** by which it can be identified; this address must be unique to allow individual access of each device. Each HP-IB device has a set of switches that are used to set its address. Thus, when a particular HP-IB device is to be accessed, it must be identified with both its interface select code and its bus address.

The interface select code is the first part of an HP-IB device selector. The interface select code of the internal HP-IB is 7; external interfaces can range from 8 to 31. The second part of an HP-IB device selector is the device's primary address, which are in the range of 0 through 30. For example, to specify the device:

the interface at select code 7	use device selector = 722
the device at primary address 22	
the interface at select code 10	use device selector = 1002
the device at primary address 2	

Remember that each device's address must be unique. The procedure for setting the address of an HP-IB device is given in the installation manual for each device. The HP-IB interface also has an address. The default address of the internal HP-IB is 21 or 20, depending on whether or not it is a System Controller, respectively. The addresses of external HP-IB interfaces are set by configuring the address switches on each interface card. Each HP-IB interface's address can be determined by reading STATUS register 3 of the appropriate interface select code, and each interface's address can be changed by writing to CONTROL register 3. See "Determining Controller Status and Address" and "Changing the Controller's Address" for further details.

## Moving Data Through the HP-IB

Data is output from and entered into the computer through the HP-IB with the OUTPUT and ENTER statements, respectively; all of the techniques described in Chapters 4 and 5 are completely applicable with the HP-IB. The only difference between the OUTPUT and ENTER statements for the HP-IB and those for other interfaces is the addressing information within HP-IB device selectors.

### Examples

```
100  Hpib=7
110  Device_addr=22
120  Device_selector=Hpib*100+Device_addr
130  !
140  OUTPUT Device_selector;"F1R7T2T3"
150  ENTER Device_selector;Reading
```

```
320  ASSIGN @Hpib_device TO 702
330  OUTPUT @Hpib_device;"Data message"
340  ENTER @Hpib_device;Number
```

```
440  OUTPUT 822;"F1R7T2T3"
```

```
380  ENTER 724;Readings(*)
```

All of the IMAGE specifiers described in Chapters 4 and 5 can also be used by OUTPUT and ENTER statements that access the HP-IB interface, and the definitions of all specifiers remain exactly as stated in those chapters.

### Examples

```
100  ASSIGN @Printer TO 701
110  OUTPUT @Printer USING "6A,3X,2D.D";Item$,Quantity

860  ASSIGN @Device TO 825
870  OUTPUT @Device USING "#,B";65,66,67,13,10
870  ENTER @Device USING "#,K";Data$
```



## Using an Interface in the HP-UX Environment

This section explains the interface locking and burst I/O, which are useful when using an interface in the HP-UX environment.

### Locking an Interface to a Process

In a multi-user environment, interface cards are usually accessible to several users. BASIC/UX supports this sharing by making no attempt to guarantee exclusive access to an interface *unless it is directed to do so*. This allows you to access instruments, for instance, on an HP-IB bus that is shared with other peripherals. Although this is not a recommended configuration, it is allowed.

BASIC/UX provides interface locking to support exclusive access to an interface. When an interface is locked to a process, all other processes are prevented from using that interface. For instance, this feature can prevent the loss of important data while a process is taking measurements from an instrument by keeping other users or processes from using the same interface.

Interface locking is enabled and disabled by using pseudo-register 255 and the interface's select code. For example:

CONTROL 7,255;1      *Enables HP-IB interface locking.*

CONTROL 7,255;0      *Disables HP-IB interface locking.*

In order to be a "good citizen" on a multi-user system, you should unlock an interface after you no longer need to have it locked.

Note that attempting to lock an HP-IB connected to a system disc will result in an error.

In addition, attempting to lock an interface that is already locked to another process will cause a program to suspend execution until:

- The interface is unlocked (by the other process to which it is currently locked).
- A timeout occurs.
- You press  or **Ctrl I/O**.



### Using the Burst I/O Mode

The default mode of HP-UX I/O transactions requires many time consuming HP-UX system calls to send data to the destination.

Another method, "burst I/O", maps the interface into your "user address space", thereby bypassing the memory buffer. This direct-write method decreases the number of calls to HP-UX I/O system routines, which establishes a short, highly tuned path for performing I/O operations. The interface is also implicitly locked when burst mode is enabled (see above explanation of interface locking).

Burst I/O provides the fastest I/O performance available with BASIC/UX for the "smaller" I/O transactions that are typical of many instruments. For instance, an 8-byte ENTER operation is over an order of magnitude faster when burst mode is enabled. For larger I/O operations, of more than 4 000 bytes for example, burst mode becomes increasingly slower than the default (buffered or DMA) I/O modes.

Burst I/O is enabled and disabled by using register 255 and the interface's select code. For example:

CONTROL 7,255;3            *Enables HP-IB interface burst I/O.*

CONTROL 7,255;0            *Disables HP-IB interface burst I/O.*

In order to be a "good citizen" on a multi-user system, you should unlock an interface after you no longer need to have it locked.

In addition, attempting to use burst mode with an interface that is already locked to another process will cause a program to suspend execution until:

- The interface is unlocked (by the other process to which it is currently locked).
- A timeout occurs.
- You press **Reset** or **Clr I/O**.

Note also that you cannot set up an ON TIMEOUT for an interface when using burst mode.

## General Structure of the HP-IB

Communications through the HP-IB are made according to a precisely defined set of rules. These rules help to ensure that only orderly communication may take place on the bus. For conceptual purposes, the organization of the HP-IB can be compared to that of a committee. A committee has certain "rules of order" that govern the manner in which business is to be conducted. For the HP-IB, these rules of order are the IEEE 488-1978 standard.

One member, designated the "committee chairman," is set apart for the purpose of conducting communications between members during the meetings. This chairman is responsible for overseeing the actions of the committee and generally enforces the rules of order to ensure the proper conduct of business. If the committee chairman cannot attend a meeting, he designates some other member to be "acting chairman."

On the HP-IB, the **System Controller** corresponds to the committee chairman. The system controller is generally designated by setting a switch on the interface and cannot be changed under program control. However, it is possible to designate an "acting chairman" on the HP-IB. On the HP-IB, this device is called the **Active Controller**, and may be any device capable of directing HP-IB activities, such as a desktop computer.

When the System Controller is first turned on or reset, it assumes the role of Active Controller. Thus, only one device can be designated System Controller. These responsibilities may be subsequently passed to another device while the System Controller tends to other business. This ability to pass control allows more than one computer to be connected to the HP-IB at the same time.

In a committee, only one person at a time may speak. It is the chairman's responsibility to "recognize" which one member is to speak. Usually, all committee members present always listen; however, this is not always the case on the HP-IB. One of the most powerful features of the bus is the ability to selectively send data to individual (or groups of) devices.

Imagine slow note takers and a fast note takers on the committee. Suppose that the speaker is allowed to talk no faster than the slowest note taker can write. This would guarantee that everybody gets the full set of notes and that no one misses any information. However, requiring all presentations to go at that slow pace certainly imposes a restriction on our committee, especially if the slow note takers do not need the information. Now, if the chairman knows which presentations are not important to the slow note takers, he can direct them to put away their notes for those presentations. That way, the speaker and the fast note taker(s) can cover more items in less time.



A similar situation may exist on the HP-IB. Suppose that a printer and a flexible disc are connected to the bus. Both devices do not need to listen to all data messages sent through the bus. Also, if all the data transfers must be slow enough for the printer to keep up, saving a program on the disc would take as long as listing the program on the printer. That would certainly not be a very effective use of the speed of the disc drive if it was the only device to receive the data. Instead, by "unlistening" the printer whenever it does not need to receive a data message, the computer can save a program as fast as the disc can accept it.

During a committee meeting, the current chairman is responsible for telling the committee which member is to be the talker and which is (are) to be the listener(s). Before these assignments are given, he must get the **attention** of all members. The talker and listener(s) are then designated, and the next data message is presented to the listener(s) by the talker. When the talker has finished the message, the designation process may be repeated.

On the HP-IB, the Active Controller takes similar action. When talker and listener(s) are to be designated, the **attention signal line (ATN)** is asserted while the talker and listener(s) are being addressed. ATN is then cleared, signaling that those devices not addressed to listen may ignore all subsequent data messages. Thus, **the ATN line separates data from commands**; commands are accompanied by the ATN line being true, while data messages are sent with the ATN line false.

On the HP-IB, devices are **addressed to talk** and **addressed to listen** in the following orderly manner. The Active Controller first sends a single command which causes all devices to **unlisten**. The talker's address is then sent, followed by the address(es) of the listener(s). After all listeners have been addressed, the data can be sent from the talker to the listener(s). Only device(s) addressed to listen accept any data that is sent through the bus (until the bus is reconfigured by subsequent addressing commands).

The data transfer, or **data message**, allows for the exchange of information between devices on the HP-IB. Our committee conducts business by exchanging ideas and information between the speaker and those listening to his presentation. On the HP-IB, **data is transferred from the active talker to the active listener(s) at a rate determined by the slowest active listener on the bus**. This restriction on the transfer rate is necessary to ensure that no data is lost by any device addressed to listen. The **handshake** used to transfer each data byte ensures that all data output by the talker is received by all active listeners.

## Examples of Bus Sequences

Most data transfers through the HP-IB involve a talker and only one listener. For instance, when an OUTPUT statement is used (by the Active Controller) to send data to an HP-IB device, the following sequence of commands and data is sent through the bus.

OUTPUT 701;"Data"

1. The unlisten command is sent.
2. The talker's address is sent (here, the address of the computer; "My Talk Address"), which is also a command.
3. The listener's address (01) is sent, which is also a command.
4. The data bytes "D", "a", "t", "a", CR, and LF are sent; all bytes are sent using the HP-IB's interlocking handshake to ensure that the listener has received each byte.

Similarly, most ENTER statements involve transferring data from a talker to only one listener. For instance, the following ENTER statement invokes the following sequence of commands and data-transfer operations.

ENTER 722;Voltage

1. The unlisten command is sent.
2. The talker's address (22) is sent, which is a command.
3. The listener's address is sent (here, the computer's address; "My Listen Address"), also a command.
4. The data is sent by device 22 to the computer using the HP-IB handshake.

Bus sequences, hardware signal lines, and more specific HP-IB operations are discussed in the "HP-IB Control Lines" and "Advanced Bus Management" sections.



## Addressing Multiple Listeners

HP-IB allows more than one device to listen simultaneously to data sent through the bus (even though the data may be accepted at differing rates). The following examples show how the Active Controller can address multiple listeners on the bus.

```
100  ASSIGN @Listeners TO 701,702,703
110  OUTPUT @Listeners;String$
120  OUTPUT @Listeners USING Image_1;Array$(*)
```

This capability allows a single OUTPUT statement to send data to several devices simultaneously. It is however, necessary for all the devices to be on the same interface. When the preceding OUTPUT statement is executed, the unlisten command is sent first, followed by the Active Controller's talk address and then listen addresses 01, 02, and 03. Data is then sent by the controller and accepted by devices at addresses 1, 2, and 3.

If an ENTER statement that uses the same I/O path name is executed by the Active Controller, the first device is addressed as the talker (the source of data) and all the rest of the devices, including the Active Controller, are addressed as listeners. The data is then sent from the device at address 01 to the devices at addresses 02 and 03 and to the Active Controller.

```
130  ENTER @Listeners;String$
140  ENTER @Listeners USING Image_2;Array$(*)
```

## Secondary Addressing

Many devices have operating modes which are accessed through the extended addressing capabilities defined in the bus standard. Extended addressing provides for a second address parameter in addition to the primary address. Examples of statements that use extended addressing are as follows.

```
100  ASSIGN @Device TO 72205 ! 22=primary, 05=secondary.
110  OUTPUT @Device;Message$

200  OUTPUT 72205;Message$

150  ASSIGN @Device TO 7220529 ! Additional secondary
160                                ! address of 29.
170  OUTPUT @Device;Message$

120  OUTPUT 7220529;Message$
```

The range of secondary addresses is 00-31; up to six secondary addresses may be specified (a total of 15 digits including interface select code and primary address). Refer to the device's operating manual for programming information associated with the extended addressing capability. The HP-IB interface also has a mechanism for detecting secondary commands. For further details, see the discussion of interrupts.

---

## General Bus Management

The HP-IB standard provides several mechanisms that allow managing the bus and the devices on the bus. Here is a summary of the statements that invoke these control mechanisms.

**ABORT** is used to abruptly terminate all bus activity and reset all devices to power-on states.

**CLEAR** is used to set all (or only selected) devices to a pre-defined, device-dependent state.

**LOCAL** is used to return all (or selected) devices to local (front-panel) control.

**LOCAL LOCKOUT** is used to disable all devices' front-panel controls.

**PPOLL** is used to perform a parallel poll on all devices (which are configured and capable of responding).

**PPOLL CONFIGURE** is used to setup the parallel poll response of a particular device.

**PPOLL UNCONFIGURE** is used to disable the parallel poll response of a device (or all devices on an interface).

**REMOTE** is used to put all (or selected) devices into their device-dependent, remote modes.

**SEND** is used to manage the bus by sending explicit command or data messages.

**SPOLL** is used to perform a serial poll of the specified device (which must be capable of responding).

**TRIGGER** is used to send the trigger message to a device (or selected group of devices).

These statements (and functions) are described in the following discussion. However, the actions that a device takes upon receiving each of the above commands are, in general, different for each device. Refer to a particular device's manuals to determine how it will respond. Detailed descriptions of the actual sequence of bus messages invoked by these statements are contained in "Advanced Bus Management" later in this chapter.

## **Remote Control of Devices**

Most HP-IB devices can be controlled either from the front panel or from the bus. If the device's front-panel controls are currently functional, it is in the Local state. If it is being controlled through the HP-IB, it is in the Remote state. Pressing the front-panel "Local" key will return the device to Local (front-panel) control, unless the device is in the Local Lockout state (described in a subsequent discussion).

The Remote message is automatically sent to all devices whenever the System Controller is powered on, reset, or sends the Abort message. A device also enters the Remote state automatically whenever it is addressed. The REMOTE statement also outputs the Remote message, which causes all (or specified) devices on the bus to change from local control to remote control. The computer must be the System Controller to execute the REMOTE statement.

### **Examples**

```
REMOTE 7
```

```
ASSIGN @Device TO 700  
REMOTE @Device
```

```
REMOTE 700
```



## Locking Out Local Control

The Local Lockout message effectively locks out the "local" switch present on most HP-IB device front panels, preventing a device's user from interfering with system operations by pressing buttons and thereby maintaining system integrity. As long as Local Lockout is in effect, no bus device can be returned to local control from its front panel.

The Local Lockout message is sent by executing the LOCAL LOCKOUT statement. This message is sent to all device on the specified HP-IB interface, and it can only be sent by the computer when it is the Active Controller.

### Examples

```
ASSIGN @Hpib TO 7  
LOCAL LOCKOUT @Hpib
```

```
LOCAL LOCKOUT 7
```

The Local Lockout message is cleared when the Local message is sent by executing the LOCAL statement. However, executing the ABORT statement does not cancel the Local Lockout message.

## Enabling Local Control

During system operation, it may be necessary for an operator to interact with one or more devices. For instance, an operator might need to work from the front panel to make special tests or to troubleshoot. And, in general, it is good systems practice to return all devices to local control upon conclusion of remote-control operations. Executing the LOCAL statement returns the specified devices to local (front-panel) control. The computer must be the Active Controller to send the LOCAL message.

### Examples

```
ASSIGN @Hpib TO 7  
LOCAL @Hpib
```

```
ASSIGN @Device TO 700  
LOCAL @Device
```

If primary addressing is specified, the Go-to-Local message is sent only to the specified device(s). However, if only the interface select code is specified, the Local message is sent to all devices on the specified HP-IB interface and any previous Local Lockout message (which is still in effect) is automatically cleared. The computer must be the System Controller to send the Local message (by specifying only the interface select code).



## Triggering HP-IB Devices

The TRIGGER statement sends a Trigger message to a selected device or group of devices. The purpose of the Trigger message is to initiate some device-dependent action; for example, it can be used to trigger a digital voltmeter to perform its measurement cycle. Because the response of a device to a Trigger Message is strictly device-dependent, neither the Trigger message nor the interface indicates what action is initiated by the device.

### Examples

```
ASSIGN @Hpib TO 7  
TRIGGER @Hpib
```

```
ASSIGN @Device TO 707  
TRIGGER @Device
```

Specifying only the interface select code outputs a Trigger message to all devices currently addressed to listen on the bus. Including device addresses in the statement triggers only those devices addressed by the statement. The computer can also respond to a trigger from another controller on the bus. See "Interrupts While Non-Active Controller" for details.

## Clearing HP-IB Devices

The CLEAR statement provides a means of "initializing" a device to its predefined, device-dependent state. When the CLEAR statement is executed, the Clear message is sent either to all devices or to the specified device(s), depending on the information contained within the device selector. If only the interface select code is specified, all devices on the specified HP-IB interface are cleared. If primary-address information is specified, the Clear message is sent only to the specified device. Only the Active Controller can send the Clear message.

### Examples

```
ASSIGN @Hpib TO 7  
CLEAR @Hpib
```

```
ASSIGN @Device TO 700  
CLEAR @Device
```

## Aborting Bus Activity

This statement may be used to terminate **all** activity on the bus and return all the HP-IB interfaces of all devices to a reset (or power-on) condition. Whether this affects other modes of the device depends on the device itself. The computer must be either the active or the system controller to perform this function. If the System Controller (which is not the current Active Controller) executes this statement, it regains active control of the bus. **Only the interface select code may be specified**; device selectors which contain primary-addressing information (such as 724) may not be used.

### Examples

```
ASSIGN @HpiB TO 7  
ABORT @HpiB  
  
ABORT 7
```

## HP-IB Service Requests

Most HP-IB devices, such as voltmeters, frequency counters, and spectrum analyzers, are capable of generating a “service request” when they require the Active Controller to take action. Service requests are generally made after the device has completed a task (such as making a measurement) or when an error condition exists (such as a printer being out of paper). The operating and/or programming manuals for each device describe the device’s capability to request service and conditions under which the device will request service.

To request service, the device sends a Service Request message (SRQ) to the Active Controller. The mechanism by which the Active Controller detects these requests is the SRQ interrupt. Interrupts allow an efficient use of system resources, because the system may be executing a program until interrupted by an event’s occurrence. If enabled, the external event initiates a program branch to a routine which “services” the event (executes remedial action).

Chapter 7 described interrupt events in general. This chapter describes the two types of interrupts that can occur on an HP-IB Interface: SRQ interrupts from external devices (that can occur while the computer is an Active Controller), and interrupts that can occur while the computer is a non-Active Controller. The first type of interrupts are described in this section. The second type are described in the section called “The Computer as a Non-Active Controller.”

### Setting Up and Enabling SRQ Interrupts

In order for an HP-IB device to be able to initiate a service routine in the Active Controller, two prerequisites must be met: the SRQ interrupt event must have a service routine defined, and the SRQ interrupt must be enabled to initiate the branch to the service routine. The following program segment shows an example of setting up and enabling an SRQ interrupt.

```
100  Hpib=7
110  ON INTR Hpib GOSUB Service_routine
120  !
130  Mask=2
140  ENABLE INTR Hpib;Mask
```

The value of the mask in the ENABLE INTR statement determines which type(s) of interrupts are to be enabled. The value of the mask is automatically written into the HP-IB interfaces's interrupt-enable register (CONTROL register 4) when this statement is executed. Bit 1 is set in the preceding example, enabling SRQ interrupts to initiate a program branch. Reading STATUS register 4 at this point would return a value of 2.

When an SRQ interrupt is generated by any device on the bus, the program branches to the service routine when the current line is exited (either when the line's execution is finished or when the line is exited by a call to a user-defined function). The service routine, in general, must perform the following operations:

- determine which device(s) are requesting service (parallel poll)
- determine what action is requested (serial poll)
- clear the SRQ line
- perform the requested action
- re-enable interrupts
- return to the former task (if applicable)

### Servicing SRQ Interrupts

The SRQ is a level-sensitive interrupt; in other words, if an SRQ is present momentarily but does not remain long enough to be sensed by the computer, an interrupt will not be generated. The level-sensitive nature of the SRQ line also has further implications, which are described in the following paragraphs.



## Example

Assume only one device is currently on the bus. The following service routine first serially polls the device requesting service, thereby clearing the interrupt request. In this case, the computer did not have to determine which device was requesting service because only one device is on the bus. It is also assumed that only service request interrupts have been enabled; therefore, the type of interrupt need not be determined either. The service is then performed, and the SRQ event is re-enabled to generate subsequent interrupts.

```
500  Serv_rtn:  Ser_poll=SPOLL(@Device)
510              ENTER @Device;Value
520              PRINT Value
530              ENABLE INTR 7 ! Use previous mask.
540              RETURN
```

The IEEE standard has defined that when an interrupting device is serially polled, it is to stop interrupting until a new condition arises (or the same condition arises again). In order to "clear" the SRQ line, it is necessary to perform a serial poll on the device. This poll is an acknowledgement from the controller to the device that it has seen the request for service and is responding. The device then removes its request for service (by releasing SRQ).

Had the SRQ line not been released, the computer would have branched to the service routine immediately upon re-enabling interrupts on this interface. This is another implication of the level-sensitive nature of the SRQ interrupt.

It is also important to note that once an interrupt is sensed and logged, the interface cannot generate another interrupt until the initial interrupt is serviced. The computer disables all subsequent interrupts from an interface until a pending interrupt is serviced. For this reason, it was necessary to allow for subsequent branching.



## Polling HP-IB Devices

The Parallel Poll is the fastest means of gathering device status when several devices are connected to the bus. Each device (with this capability) can be programmed to respond with one bit of status when Parallel Polled, making it possible to obtain the status of several devices in one operation. If a device responds affirmatively ("I need service") to a Parallel Poll, then more information as to its specific status can be obtained by conducting a Serial Poll of the device.

### Configuring Parallel Poll Responses

Certain devices can be remotely programmed by the Active Controller to respond to a Parallel Poll. A device which is currently configured for a Parallel Poll responds to the poll by placing its current status on one of the bus data lines. The logic sense of the response and the data-bit number can be programmed by the PPOLL CONFIGURE statement. No multiple listeners can be specified in the statement; if more than one device is to respond on a single bit, each device must be configured with a separate PPOLL CONFIGURE statement.

#### Example

```
ASSIGN @Device TO 701
PPOLL CONFIGURE @Device;Configure_code
```

The value of `Configure_code` (any numeric expression can be specified) is first rounded to an integer and then used to configure the device's Parallel Poll Response. The least-significant 3 bits (2 thru 0) of the expression are used to determine which data line the device is to respond on (place its status on). Bit 3 specifies the logic sense of the Parallel Poll Response bit of the device. For instance, a value of 0 implies that the device's response is 0 when its Status Bit message is "I need service."

#### Example

The following statement configures the device at address 01 on the HP-IB interface at select code 7 to respond by placing a 0 on bit 4 (DIO5) when its Status Bit response is affirmative.

```
PPOLL CONFIGURE 701; 4
```

### **Conducting a Parallel Poll**

The PPOLL function returns a single byte containing up to 8 status bit messages of the devices on the bus (which are capable of responding to the Parallel Poll. Each bit returned by the function corresponds to the status bit of the device(s) configured to respond to the Parallel Poll. (Recall that one or more devices can respond on a single line.) The PPOLL function can only be executed by the Active Controller.

#### **Example**

```
Response=PPOLL(7)
```

### **Disabling Parallel Poll Responses**

The PPOLL UNCONFIGURE statement gives the Active Controller the capability of disabling the Parallel Poll responses of one or more devices on the bus.

#### **Examples**

```
PPOLL UNCONFIGURE 705
```

The following statement disables all devices on the HP-IB interface at select code 8 from responding to a Parallel Poll.

```
PPOLL UNCONFIGURE 8
```

If no primary addressing is specified, all bus devices are disabled from responding to a Parallel Poll. If primary addressing is specified, only the specified devices (which have the Parallel Poll Configure capability) are disabled.

### **Conducting a Serial Poll**

A sequential poll of individual devices on the bus is known as a Serial Poll. One entire byte of device-specific status is returned in response to a Serial Poll. This byte is called the "Status Byte" message and, depending on the device, may indicate an overload, a request for service, or a printer being out of paper. The particular response of each device depends on the device.

The SPOLL function performs a Serial Poll of the specified device; the computer must currently be the Active Controller in order to execute this function.

## Examples

```
ASSIGN @Device TO 700  
Status_byte=SPOLL(700)
```

```
Spoll_724=SPOLL(724)
```

Just as the Parallel Poll is not defined for individual devices, the Serial Poll is meaningless for an interface; therefore, **primary addressing must be used** with the SPOLL function.

---

## Advanced Bus Management

Bus communication involves both sending data to devices and sending commands to devices and the interface itself. "General Structure of the HP-IB" stated that this communication must be made in an orderly fashion and presented a brief sketch of the differences between data and commands. However, most of the bus operations described so far in this chapter involve sequences of commands and/or data which are sent automatically by the computer when HP-IB statements are executed. This section describes both the commands and data sent by HP-IB statements and how to construct your own, custom bus sequences.

### The Message Concept

The main purpose of the bus is to send information between two (or more) devices. These quantities of information sent from talker to listener(s) can be thought of as messages. However, before data can be sent through the bus, it must be properly configured. A sequence of commands is generally sent before the data to inform bus devices which is to send and which is (or are) to listen to the subsequent message(s). These commands can also be thought of as messages.

Most bus messages are transmitted by sending a byte (or sequence of bytes) with numeric values of 0 through 255 through the bus data lines. When the Attention line (ATN) is true, these bytes are considered commands; when ATN is false, they are interpreted as data. Bus command groups and their ASCII characters and codes are shown in "Bus Commands and Codes".



## Types of Bus Messages

The messages can be classified into twelve types. This computer is capable of implementing all twelve types of interface messages. The following list describes each type of message.

1. A Data message consists of information which is sent from the talker to the listener(s) through the bus data lines.
2. The Trigger message causes the listening device(s) to initiate device-dependent action(s).
3. The Clear message causes either the listening device(s) or all of the devices on the bus to return to their device-dependent "clear" states.
4. The Remote message causes listening devices to change to remote program control when addressed to listen.
5. The Local message clears the Remote message from the listening device(s) and returns the device(s) to local front-panel control.
6. The Local Lockout message disables a device's front-panel controls, preventing a device's operator from manually interfering with remote program control.
7. The Clear Lockout/Local message causes all devices on the bus to be removed from Local Lockout and to revert to the Local state. This message also clears the Remote message from all devices on the bus.
8. The Service Request message can be sent by a device at any time to signify that the device needs to interact with the the Active Controller. This message is cleared by sending the device's Status Byte message, if the device no longer requires service.
9. A Status Byte message is a byte that represents the status of a single device on the bus. This byte is sent in response to a serial poll performed by the Active Controller. Bit 6 indicates whether the device is sending the Service Request message, and the remaining bits indicate other operational conditions of the device.
10. A Status Bit message is a single bit of device-dependent status. Since more than one device can respond on the same line, this Status Bit may be logically combined and/or concatenated with Status Bit messages from many devices. Status Bit messages are returned in response to a Parallel Poll conducted by the Active Controller.
11. The Pass Control message transfers the bus management responsibilities from the Active Controller to another controller.

12. The Abort message is sent by the System Controller to assume control of the bus unconditionally from the Active Controller. This message terminates all bus communications, but is not the same as the Clear message.

These messages represent the full implementation of all HP-IB system capabilities; all of these messages can be sent by this computer. However, each device in a system may be designed to use only the messages that are applicable to its purpose in the system. It is important for you to be aware of the HP-IB functions implemented on each device in your HP-IB system to ensure its operational compatibility with your system.

## Bus Commands and Codes

The following table shows the decimal values of IEEE-488 command messages. Remember that **ATN is true** during all of these commands. Notice also that these commands are separated into four general categories: Primary Command Group, Listen Address Group, Talk Address Group, and Secondary Command Group. Subsequent discussions further describe these commands.

**Table 12-1. HP-IB Commands and Codes**

Decimal Value	ASCII Character	Interface Message	Description
		<b>PCG</b>	<b>Primary Command Group</b>
1	SOH	GTL	Go to Local
4	EOT	SDC	Selected Device Clear
5	ENQ	PPC	Parallel Poll Configure
8	BS	GET	Group Execute Trigger
9	HT	TCT	Take Control
17	DC1	LLO	Local Lockout
20	DC4	DCI	Device Clear
21	NAK	PPU	Parallel Poll Unconfigure
24	CAN	SPE	Serial Poll Enable
25	EM	SPD	Serial Poll Disable
		<b>LAG</b>	<b>Listen Address Group</b>
32-62	Space through > (Numbers & Special Chars.)		Listen Addresses 0 through 30
63	?	UNL	Unlisten
		<b>TAG</b>	<b>Talk Address Group</b>
64-94	@ through ↑ (Uppercase Letters)		Talk Addresses 0 through 30
95	— (underscore)	UNT	Untalk
		<b>SCG</b>	<b>Secondary Command Group</b>
96-126	‘ through ~ (Lowercase Letters)		Secondary Commands 0 through 30
127	DEL		Ignored



### Address Commands and Codes

The following table shows the ASCII characters and corresponding codes of the Listen Address Group and Talk Address Group commands. The next section describes how to send these commands.

**Table 12-2. HP-IB Listen and Talk Address Commands**

Listen Address Character	Talk Address Character	Address Code	Address Switch Settings
Space	@	0	0 0 0 0 0
!	A	1	0 0 0 0 1
"	B	2	0 0 0 1 0
#	C	3	0 0 0 1 1
\$	D	4	0 0 1 0 0
%	E	5	0 0 1 0 1
&	F	6	0 0 1 1 0
'	G	7	0 0 1 1 1
(	H	8	0 1 0 0 0
)	I	9	0 1 0 0 1
*	J	10	0 1 0 1 0
+	K	11	0 1 0 1 1
,	L	12	0 1 1 0 0
-	M	13	0 1 1 0 1
.	N	14	0 1 1 1 0
/	O	15	0 1 1 1 1
0	P	16	1 0 0 0 0
1	Q	17	1 0 0 0 1
2	R	18	1 0 0 1 0

**Table 12-2. HP-IB Listen and Talk Address Commands (continued)**

Listen Address Character	Talk Address Character	Address Code	Address Switch Settings
3	S	19	1 0 0 1 1
4	T	20	1 0 1 0 0
5	U	21	1 0 1 0 1
6	V	22	1 0 1 1 0
7	W	23	1 0 1 1 1
8	X	24	1 1 0 0 0
9	Y	25	1 1 0 0 1
:	Z	26	1 1 0 1 0
;	[	27	1 1 0 1 1
<	/	28	1 1 1 0 0
=	]	29	1 1 1 0 1
>	^	30	1 1 1 1 0

The preceding table implicitly shows that:

- Listen address commands can be calculated from the primary address by using one of the following equations:

`Listen_address=32+Primary_address`

or

`Listen_address$=CHR$(32+Primary_address)`

- Similarly, talk address commands can be calculated from the primary address by using one of the following equations

`Talk_address=64+Primary_address`

or

`Talk_address$=CHR$(64+Primary_address)`

However, the table does not show that:

- the Unlisten command is "?", CHR\$(63)
- the Untalk command is "\_", CHR\$(95)
- therefore, primary address 31 is an unusable device address, but can be used to send the Unlisten and Untalk commands.

## Explicit Bus Messages

It is often desirable (or necessary) to manage the bus by sending explicit sequences of bus messages. The SEND statement is the vehicle by which explicit commands and data can be sent through the bus. The SEND statement is also a method of sending data with odd parity through the bus (instead of using the PARITY attribute discussed in the "I/O Path Attributes" chapter). This section shows several uses of this statement.

### Examples of Sending Commands

As a simple example, suppose the following statement is executed by the Active Controller to configure the bus (i.e., to address the talker and listener).

```
OUTPUT 701 USING "#,K"
```

The SEND statement can be used to send the same sequence of commands, as shown in the following statement.

```
SEND 7;CMD "?U!"
```

This statement configures the bus explicitly by sending the following commands:

- the unlisten command (ASCII character "?"; decimal code 63)
- talk address 21 (ASCII character "U"; decimal code 85)
- listen address 1 (ASCII character "!"; decimal code 33)

The same sequence of commands and data is sent with any of the following statements.

```
SEND 7;CMD UNL MTA LISTEN 1
```

```
SEND 7;CMD UNL TALK 21 LISTEN 1
```

```
SEND 7;CMD 32+31,64+21,32+1
```



Commands can be sent by specifying the secondary keyword CMD. The list of commands (following CMD) can be any numeric or string expressions. If more than one expression is listed, they must be separated by commas. A numeric expression will be evaluated, rounded to an integer (MOD 256), and sent as one byte. Each character of a string expression will be sent individually. **All bytes are sent with ATN true.** The computer must be the current Active Controller to send commands.

```
SEND 1sc;CMD 8           ! Group Execute Trigger
SEND 1sc;TALK New_controller CMD 9 ! Pass Control
SEND 8;CMD 1             ! Go to Local
```

If SEC is used, the specified secondary commands will be sent. An extended talker may be addressed by using SEC after the talk address; extended listener(s) may be addressed by using SEC after the listen address(es).

```
SEND 7;MTA UNL LISTEN 1 CMD 5 SEC 16 ! SEND PPD.
```

The computer must be the Active Controller to send CMD, LISTEN, UNL, MLA, TALK, UNT, MTA, and SEC. If a non-Active Controller attempts to send any of these messages, an error is reported.

Simulate the following SPOLL function with SEND and ENTER statements.

```
A=SPOLL(724)
```

When an SPOLL is performed, the resulting bus activity is:

- Unlisten command
- My Listen Address (the computer's listen address; MLA)
- device's talk address (one of the TAG commands)
- Serial Poll Enable command (SPE; decimal code 24)
- one data byte is read (the Status Byte message)
- Serial Poll Disable (SPD; decimal code 25)
- Untalk command

This is accomplished by either of the following sequences:

```
SEND 7;CMD "?5X"&CHR$(24)    ! Configure the bus; send SPE.  
ENTER 7 USING "#,B";A        ! Read Status Byte.  
SEND 7;CMD CHR$(25)&"_"      ! Send SPD and Untalk.
```

```
SEND 7;UNL MLA TALK 24 CMD 24  
ENTER 7 USING "#,B";A  
SEND 7;CMD 25 UNT
```

The preceding secondary keywords provide the capability of sending various command messages through the bus. The activity that results on the bus when several other high-level commands are issued is summarized in "HP-IB Message Mnemonics".

### Examples of Sending Data

Data messages can be sent by specifying the secondary keyword DATA. If the computer is the Active Controller, the data is sent immediately. However, if the computer is not the Active Controller, it waits to be addressed to talk before sending the data.

```
SEND 7;DATA "Message",13,10  ! Send with CR/LF.
```

```
SEND Bus;DATA "Data" END      ! Send with EOI.
```

The data list may contain any mixture of numeric or string expressions; if more than one expression is specified, they must be separated by commas. Each numeric expression is evaluated as an integer (MOD 256) and sent as a single byte. Each string item is evaluated and all resultant characters are sent serially. Each byte is **sent with ATN false** (sent as a data message). The last expression may be followed by the secondary keyword END, which causes the EOI terminator to be sent concurrently with the last data byte.

As another example, simulate this ENTER statement with a SEND statement.

```
ENTER 724;Number,String$
```

Any of the following pairs of statements can be used to accomplish the same operation.

```
SEND 7;UNL TALK 24 MLA  
ENTER 7;Number,String$
```

```
SEND 7;UNL TALK 24 LISTEN 21  
ENTER 7;Number,String$
```

```
SEND 7;CMD "?X5"  
ENTER 7;Number, String$
```

## HP-IB Message Mnemonics

This section contains the descriptions of several bus messages described by the IEEE 488-1978 standard. The following table describes message mnemonics, their meanings, and the secondary keywords used with the SEND statement. The HP-IB messages that require primary keywords are noted in the table.

All BASIC statements which send HP-IB messages (except SEND) always set ATN-true (command) messages with the most-significant bit set to zero. Using CMD (with SEND) allows you to send ATN-true messages with the most-significant bit set to one. This may be useful for non-standard IEEE-488 devices which require the most-significant bit to have a particular value.

The CMD and DATA secondary keywords of SEND statements allow string expressions as well as numeric expressions (e.g., CMD "?" is the same as CMD 63). All other secondary keywords which need data require **numeric** expressions. Keep this in mind while reading through this table.

**Table 12-3. HP-IB Messages and Mnemonics**

Message Mnemonic	Message Description	SEND Clause Required (numeric values are decimal)
DAB	Data Byte	DATA 0 through 255
DCL	Device Clear	CMD 20 (or 148)
EOI	End or Identify	DATA data list END
GET	Group Execute Trigger	CMD 8 (or 136)
GTL	Go To Local	CMD 1 (or 129)
IFC	Interface Clear	Not possible with SEND; use the ABORT statement.
LAG	Listen Address	LISTEN 0 through 30; or CMD 32 through 62; or CMD 160 through 190
LLO	Local Lockout	CMD 17
MLA	My Listen Address	MLA
MTA	My Talk Address	MTA



**Table 12-3. HP-IB Messages and Mnemonics (continued)**

<b>Message Mnemonic</b>	<b>Message Description</b>	<b>SEND Clause Required (numeric values are decimal)</b>
PPC	Parallel Poll Configure	CMD 5 (or 133)
PPD	Parallel Poll Disable	SEC 16; or CMD 112 (or 240) (Must be preceded by PPC.)
PPE	Parallel Poll Enable	SEC 0+Mask: SEC 0 through 15; or CMD 96 through 111; or CMD 224 through 239 (Must be preceded by PPC.)
PPU	Parallel Poll Unconfig.	CMD 21 (or 149)
PPOLL	Parallel Poll	Not possible with SEND; use the PPOLL function.
REN	Remote Enable	Not possible with SEND; use the REMOTE statement.
SDC	Selected Device Clear	CMD 4 (or 132)
SPD	Serial Poll Disable	CMD 25 (or 153)
SPE	Serial Poll Enable	CMD 24 (or 152)
TAD	Talk Address	TALK 0 through 30; or CMD 64 through 94; or CMD 192 through 222
TCT	Take Control	CMD 9 (or 137)
UNL	Unlisten	UNL; or LISTEN 31; or CMD 63 (or 191)
UNT	Untalk	UNT; or TALK 31; or CMD 95 (or 223)

---

## **The Computer As a Non-Active Controller**

The section called "General Structure of the HP-IB" described how communications take place through HP-IB Interfaces. The functions of the System Controller and Active Controller were likened to a "committee chairman" and "acting chairman," respectively, and the functions of each were described. This section describes how the Active Controller may "pass control" to another controller and assume the role of a non-Active Controller. This action is analogous to designating another committee member to take the responsibility of acting chairman and then becoming a committee member who listens to the acting chairman and speaks when given the floor. The following topics will be discussed:

- Determining whether the computer is currently the Active Controller and/or System Controller
- Determining the computer's HP-IB primary address, and changing it, if necessary
- Passing control to another HP-IB controller
- Requesting service from the Active Controller
- Responsibilities of being a non-Active Controller
- Responding to interrupts that occur while non-Active Controller

### **Determining Controller Status and Address**

It is often necessary to determine if an interface is the System Controller and to determine whether or not it is the current Active Controller. It is also often necessary to determine or change the interface's primary address. The example program shown in the beginning of this chapter interrogated interface STATUS registers and printed the resultant System-Controller status and primary address. Those operations are explained in the following paragraphs.

### Example

Executing the following statement reads STATUS register 3 (of the internal HP-IB) and places the current value into the variable Stat\_and\_addr. Remember that if the statement is executed from the keyboard, the variable Stat\_and\_addr must be defined in the current context.

```
STATUS 7,3;Stat_and_addr
```

### STATUS Register 3

### Controller Status and Address

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Active Controller	0	Primary Address of HP-IB Interface				
Value=128	Value=64	Value=0	Value=16	Value=8	Value=4	Value=2	Value=1

If **bit 7** is set (1), it signifies that the interface is the System Controller; if clear (0), it is not the System Controller. Only one controller on each HP-IB interface should be configured as the System Controller.

If **bit 6** is set (1), it signifies that the interface is currently the Active Controller; if it is clear (0), another controller is currently the Active Controller.

**Bits 4 through 0** represent the current value of the interface's primary address, which is in the range of 0 through 30. The power-on default value for the internal HP-IB is 21 (if it is the System Controller) and 20 (if not the System Controller). For external HP-IB interfaces, the default address is set to 21 at the factory but may be changed by setting the address switches on the card itself.

### Example

Calculate the primary address of the interface from the value previously read from STATUS register 3.

```
Intf_addr=Stat_and_addr MOD 32
```

This numerical value corresponds to the talk (or listen) address sent by the computer when an OUTPUT (or ENTER) statement containing primary-address information is executed. Talk and listen addresses are further described in "Advanced Bus Management".



## Changing the Controller's Address

It is possible to use the CONTROL statement to change an HP-IB interface's address.

### Example

```
CONTROL 7,3;Intf_addr
```

The value of Intf\_addr is used to set the address of the HP-IB interface (in this case, the internal HP-IB). The valid range of addresses is 0 through 30; **address 31 is not used**. Thus, if a value greater than 30 is specified, the value MOD 32 is used (for example: 32 MOD 32 equals 0, 33 MOD 32 equals 1, 62 MOD 32 equals 30, and so forth).

## Passing Control

The current Active Controller can pass this capability to another computer by sending the Take Control message (TCT). The Active Controller must first address the prospective new Active Controller to talk, after which the TCT message is sent. If the other controller accepts the message, it then assumes the role of Active Controller; this computer then assumes the role of a non-Active Controller.

Passing control can be accomplished in one of two ways: it can be handled by the system, or it can be handled by the program. To handle it programmatically, use the PASS CONTROL statement. For example, the following statements first define the HP-IB Interface's select code and new Active Controller's primary address and then pass control to that controller.

```
100 Hp_ib=7
110 New_ac_addr=20
120 PASS CONTROL 100*Hp_ib+New_ac_addr
```

The following statements perform the same functions.

```
100 Hp_ib=7
110 New_ac_addr=20
120 SEND Hp_ib;UNL TALK New_ac_addr CMD 9
```

Once the new Active Controller has accepted the TCT command, the controller passing control assumes the role of a non-Active Controller (or "HP-IB device") on the specified HP-IB Interface. The next section describes the responsibilities of the computer while it is a non-Active Controller.

### Restrictions to Passing Control with BASIC/UX

On BASIC/UX if the HP-IB device contains swap space or a mounted file system, you cannot pass control to that device.

### Interrupts While Non-Active Controller

When the computer is not an Active Controller, it must be able to detect and respond to many types of bus messages and events.

The computer (as a non-Active Controller) needs to keep track of the following information.

- It must keep track of itself being addressed as a listener so that it can enter data from the current active talker.
- It must keep track of itself being addressed as a talker so that it can transmit the information desired by the active controller.
- It must keep track of being sent a Clear, Trigger, Local, or Local Lockout message so that it can take appropriate action.
- It must keep track of control being passed from another controller.

One way to do this is to continually monitor the HP-IB interface by executing the STATUS statement and then taking action when the values returned match the values desired. This is obviously a great waste of computer time if the computer could be performing other tasks. Instead, the interface hardware can be enabled to monitor bus activity and then generate interrupts when certain events take place.

The computer has the ability to keep track of the occurrences of all of the preceding events. In fact, it can monitor up to 16 different interrupt conditions. STATUS registers 4, 5 and 6 provide access to the interface state and interrupt information necessary to design very powerful systems with a great degree of flexibility.

Each individual bit of STATUS register 4 corresponds to the same bit of STATUS register 5. Register 4 provides information as to which condition **caused** an interrupt, while register 5 keeps track of which interrupt conditions are **currently enabled**. To enable a combination of conditions, add the decimal values for each bit that you want set in the interrupt-enable register. This total is then used as the mask parameter in an ENABLE INTR statement.

Note that non-active controller interrupts that occur during an ENTER or OUTPUT operation are lost.

**STATUS Register 5****Interrupt Enable Mask**

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel Poll Configuration Change	My Talk Address Received	My Listen Address Received	EOI Received	SPAS	Remote/Local Change	Talker/Listener Address Change
Value=-32 768	Value=16 384	Value=8 192	Value=4 096	Value=2 048	Value=1 024	Value=512	Value=256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger Received	Handshake Error	Unrecognized Universal Command	Secondary Command While Addressed	Clear Received	Unrecognized Addressed Command	SRQ Received	IFC Received
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**Bit 15** enables an interrupt upon becoming the Active Controller. The computer then has the ability to manage bus activities.

**Bit 14**<sup>1</sup> enables an interrupt upon detecting a change in Parallel Poll Configuration.

**Bit 13** enables an interrupt upon being addressed as an active talker by the Active Controller.

**Bit 12** enables an interrupt upon being addressed as an active listener by the Active Controller.

**Bit 11** enables an interrupt when an EOI is received during an ENTER operation (the EOI signal line is also described in "HP-IB Control Lines").

**Bit 10** enables an interrupt when the Active Controller performs a Serial Poll on the computer (in response to its service request).

**Bit 9** enables an interrupt upon receiving either the Remote or the Local message from the active controller, if addressed to listen. The action taken by the computer is, of course, dependent on the user-programmed service routine.

<sup>1</sup> This condition requires accepting data from the bus and then explicitly releasing the bus. Refer to the "Advanced Bus Management" section for further details.



**Bit 8** enables an interrupt upon a change in talk or listen address. An interrupt will be generated if the computer is addressed to listen or talk or “idled” by an Unlisten or Untalk command.

**Bit 7** enables an interrupt upon receiving a Trigger message, if the computer is currently addressed to listen. This interrupt can be used in situations where the computer may be “armed and waiting” to initiate action; the active controller sends the Trigger message to the computer to cause it to begin its task.

**Bit 6** enables an interrupt if a bus error occurs during an OUTPUT statement. Particularly, the error occurs if none of the devices on the bus respond to the HP-IB’s interlocking handshake (see “HP-IB Control Lines”). The error typically indicates that either a device is not connected or that its power is off.

**Bit 5<sup>1</sup>** enables an interrupt upon receiving an unrecognized Universal Command. This interrupt condition provides the computer with the capability of responding to new definitions that may be adopted by the IEEE standards committee.

**Bit 4<sup>1</sup>** enables an interrupt upon receiving a Secondary Command (extended addressing) after the interface receives either its primary talk address or primary listen address. Again, this interrupt provides the computer with a way to detect and respond to special messages from another controller.

**Bit 3** enables an interrupt on receiving a Clear message. Reception of either a Device Clear message (to all devices) or a Selected Device Clear message (addressed to the computer) will cause this type of interrupt. The computer is free to take any “device-dependent” action; such as, setting up all default values again, or even restarting the program, if that is defined by the programmer to be the “cleared” state of the machine.

**Bit 2<sup>1</sup>** enables an interrupt upon receiving an unrecognized Addressed Command, if the computer is currently addressed to listen. This interrupt is used to intercept and respond to bus commands which are not defined by the standard.

**Bit 1** enables an interrupt upon detecting a Service Request.

**Bit 0** enables an interrupt upon detecting an Interface Clear (IFC). The interrupt is generated only when the computer is not the System Controller, as only a System Controller is allowed to set the Interface Clear signal line. The service routine typically is used to recover from the abrupt termination of an I/O operation caused by another controller sending the IFC message.

---

<sup>1</sup> This condition requires accepting data from the bus and then explicitly releasing the bus. Refer to the “Advanced Bus Management” section for further details.

Note that most of the conditions are state- or event-sensitive; the exception is the SRQ event, which is level-sensitive. State- or event-sensitive events can never go unnoticed by the computer as can service requests; the event's occurrence is "remembered" by the computer until serviced.

For instance, if the computer is enabled to generate an interrupt on becoming addressed as a talker, it would interrupt the first time it received its own talk address. After having responded to the service request (most likely with some sort of OUTPUT operation), it would not generate another interrupt, even if it was still left assigned as a talker by the Active Controller. Thus, it would not generate another interrupt until the event occurred a second time.

An oversimplified example of a service routine that is to respond to multiple conditions might be as follows.

```
100  ON INTR Hpib GOSUB Service
110  Mask=INT(2^13)+INT(2^12)
120  ENABLE INTR Hpib;Mask ! Interrupt on receiving
130                                ! talk or listen addr.
140  Idle: GOTO Idle
150      !
160  Service: STATUS Hpib,4;Status,Mask
170          IF BIT(Status,13) THEN Talker
180          IF BIT(Status,12) THEN Listener
190          RETURN! Ignore other interrupts.
200  Talker: ! Take action for talker.
210          GOTO Exit_point
220      !
230  Listener: ! Take action for listener.
240      !
250  Exit_point: ENABLE INTR Hpib;Mask
260              RETURN
270  END
```

Register 4, the interrupt status register, is a "read-destructive" register; reading the register with a STATUS statement returns its contents and then **clears the register** (to a value of 0). If the service routine's action depends on the contents of STATUS register 4, the variable in which it is stored must not be used for any other purposes before all of the information that it contains has been used by the service routine.

Register 4, the interrupt status register, is a “read-destructive” register; reading the register with a STATUS statement returns its contents and then **clears the register** (to a value of 0). If the service routine’s action depends on the contents of STATUS register 4, the variable in which it is stored must not be used for any other purposes before all of the information that it contains has been used by the service routine.

The computer is automatically addressed to talk (by the Active Controller) whenever it is Serially Polled. If interrupts are concurrently enabled for My Address Change and/or Talker Active, the ON INTR branch will be initiated due to the reception of the computer’s talk address. However, since the Serial Poll is automatically finished with the Untalk Command, the computer may no longer be addressed to talk by the time the interrupt service routine begins execution. See “Responding to Serial Polls” for further details.

### Addressing a Non-Active Controller

The bus standard states that a **non-Active Controller cannot perform any bus addressing**. When **only the interface select code** is specified in an ENTER or OUTPUT statement that uses an HP-IB interface, **no bus addressing is performed**.

If the computer currently is **not the Active Controller**, it can still act as either talker or listener, provided it has been **previously addressed** as such. Thus, if an ENTER or OUTPUT statement is executed while the computer is not an Active Controller, the computer first determines whether or not it is an active talker or listener. If not addressed to talk or listen, the computer waits until it is properly addressed and then finishes executing the statement. It relies on the Active Controller (another computer or device) to perform the bus addressing, and then simply participates as a device in the exchange of the data. Example statements which send and receive data while the computer is not an Active Controller are as follows.

```
100 OUTPUT 7;"Data" ! If not talker, then wait until
110                  ! addressed as talker to send data.

200 ENTER 7;Data$   ! If not listener, then wait until
210                  ! addressed as listener to accept data.
```



If the computer is the **Active Controller**, it proceeds with the data transfer without addressing which devices are talker and listener(s). However, if the bus has not been configured previously, an error is reported (**Error 170 I/O operation not allowed**). The following program does not require the "overhead" of addressing talker and listeners each time the OUTPUT statement in the FOR..NEXT loop is executed, because the bus is not reconfigured each time.

```
100  OUTPUT 701 USING "#,K"  ! Configure the bus:
110                                ! This interface = talker, and
120                                ! printer (701) = listener.
130                                !
140  FOR Iteration=1 TO 25
150      OUTPUT 7;"Data message"
160  NEXT Iteration
170  !
180  END
```

This type of HP-IB addressing should be used with the understanding that if an event initiates a branch between the time that the initial addressing was made (line 100) and the time that any of the OUTPUT statements are executed (line 150), the event's service routine may reconfigure the bus differently than the initial configuration. If so, the data will be directed to the device(s) addressed to listen by the last I/O statement executed in the service routine. Events may need to be disabled if this method of addressing is used.

In general, most applications do not require this type of bus-overhead minimization; the computer's I/O language has already been optimized to provide excellent performance. Advanced methods of explicit bus management will be described in the section called "Advanced Bus Management".

---

#### Note

This type of HP-IB addressing is not allowed for TRANSFER in BASIC/UX. If only a select code is specified, and that select code is Active Controller, an error is reported (**Error 170 I/O operation not allowed**).

---

## Requesting Service

When the computer is a non-Active Controller, it has the capability of sending an SRQ to the current Active Controller. The following statement is an example of requesting service from the Active Controller of the HP-IB Interface on select code 7.

CONTROL 7,1;64

The REQUEST statement can be used to perform the same function.

REQUEST 7;64

Both of the preceding example place a logic True on the SRQ line. (Note that the line may already be set True by another device.) Other bits may be set in the Status Byte message, indicating that other device-dependent conditions exist.

The SRQ line is held True until the Active Controller executes a Serial Poll or this computer executes a REQUEST with bit 6 equal to 0. (Note also that the line may still be held True by another device.)

When the Active Controller detects an SRQ message, it usually polls device(s) on the bus to determine which need(s) service and what kind of service is needed. To determine *which* device(s) are requesting service, the Active Controller conducts a Parallel Poll. If there are not more than one device currently capable of requesting service, the Parallel Poll is not necessary.

The Parallel Poll is conducted by sending an Identify (ATN & EOI). This non-Active Controller's response to a Parallel Poll performed by the Active Controller depends on the current Parallel Poll Response set up for this controller. Setting up this controller's Parallel Poll Response is described in the next section.

If the Active Controller needs to determine what service action is required for a particular device, it performs a Serial Poll on the device(s) that responded to the Parallel Poll with an "I need service." As each device is Serially Polled, it responds by placing its Status Byte on the bus.

This non-Active Controller's response to a Serial Poll performed by the Active Controller is handled automatically by the system. The Status Byte is the byte sent to the Serial Poll Response Byte Register (with CONTROL or REQUEST, as shown above). A subsequent section further describes this non-Active Controller's responses to Serial Polls.

## Responding to Parallel Polls

Before performing a Parallel Poll of bus devices, the Active Controller configures selected device(s) to respond on one of the eight data lines. Each device is directed to respond on a particular data line with a logic True or False; the logic sense of the response informs the Active Controller either "I do need service" or "I don't need service." The logic sense of the response is also specified by the Active Controller. This response to the Parallel Poll is known as the Status Bit message.

After the desired devices have been told how to respond, the Active Controller can send the Identify message and read the Status Bits placed on the data lines to determine which device(s) need service. Identify is sent by placing ATN and EOI in the logic True state. All devices which are currently configured for the poll respond as configured.

To configure its own Parallel Poll Response, the computer must receive a Parallel Poll Configure (PPC) command followed by a Parallel Poll Enable (PPE) command from the Active Controller. Receiving this "Parallel Poll Configuration Change" generates an interrupt (this type of interrupt is enabled by setting bit 14 of the Interrupt Enable Register). The service routine takes care of configuring this controller's response by first accepting the encoded "configure byte" (the PPE command from the Active Controller) and then setting up a corresponding response.

The desired Status Bit message can be configured and sent by one of two methods. The first, and simplest, method is to define an automatic response by using the PPOLL RESPONSE statement. With this method, the computer reads the configure byte from the data lines (HP-IB STATUS Register 7) and then writes the byte's numeric value into HP-IB CONTROL Register 5. The following statements show an example of configuring this controller's Parallel Poll Response.

```
100 STATUS 7,7;Configure_code
110 CONTROL 7,5;Configure_code
120 I_need_service=0
130 PPOLL RESPONSE 7;I_need_service
```

When the computer receives a subsequent Identify from the Active Controller, the specified response ("I do/don't need service") is automatically sent to the Active Controller. The computer will probably need to respond to a Serial Poll, which is described in the next section.

The second method requires that the service routine decode the configure byte and set up the corresponding response. The configure byte read from HP-IB STATUS Register 7 contains 5 bits of data encoded with the following information:



**CONTROL Register 5****Parallel Poll Response Mask**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used			Uncon- figure	Logic Sense	Data Bit Used for Response		
Value=128	Value=64	Value=0	Value=16	Value=8	Value=4	Value=2	Value=1

**Bit 4** determines whether a response will or will not be configured. A 1 tells this controller *not* to configure a response, and a 0 tells the controller to configure a response.

**Bit 3** determines the logic sense of the Status Bit. If this bit is 0, then the "I need service" message is a 0; if this bit is 1, the "I need service" message is 1.

**Bits 2 through 0** determine the data line on which the Status Bit is to be placed. For instance, if these bits are "000", then the Status Bit is to be placed on DIO1. If these bits are "111", then the response is to be placed on DIO8.

The service routine calculates the desired response and places the appropriate bit pattern in HP-IB CONTROL Register 2. For instance, if the configure byte has a value of 12 (positive-true logic on DIO5 for "I need service"), the value sent to CONTROL Register 2 is 16 for "I need service." The appropriate statement might be:

```
CONTROL 7,2;16
```

When the Identify is received from the Active Controller, the specified response is made automatically.

As another example, suppose that the configure byte has a value of 7. The Status Bit to be written into DIO8 would be a 0 for "I need service." The corresponding statement might be:

```
CONTROL 7,2;0
```

The following general routine calculates the value to be sent to CONTROL Register 2:

```
790 STATUS 7,7;Config_code ! Read data lines.
800 Config_code=Config_code MOD 256 ! Strip 8 MSBs.
810 Unconfig=BIT(Config_code,4)
820 Sense=BIT(Config_code,3)
830 IF Unconfig=1 OR Sense=0 THEN ! Unconfigure.
840   Ppoll_response=0
850 ELSE ! Configure.
860   Status_bit=Config_code MOD 8 ! Get bits 2-0.
870   Ppoll_response=2^Status_bit ! Set proper bit.
880 END IF
890 CONTROL 7,2;Ppoll_response
```

## Responding to Serial Polls

As a non-Active Controller, the response to Serial Polls is automatically handled by the system. The desired Serial Poll Response Byte is sent to HP-IB CONTROL Register 1. If bit 6 is set (bit 6 has a value of 64), an SRQ is indicated from this controller. All other bits can be considered to be "device-dependent," and can thus be set according to the program's needs.

The following statement sets up a response with SRQ and bits 1 and 0 set to 1's.

```
CONTROL 7,1;64+2+1
```

When the Active Controller performs a Serial Poll on this non-Active Controller, the specified byte is automatically sent to the Active Controller by the system.

This non-Active Controller is automatically addressed to talk by the Active Controller during a Serial Poll. If interrupts are concurrently enabled for My Address Change and/or Talker Active interrupts, the ON INTR branch will be initiated due to the reception of this controller's talk address. However, since the Serial Poll Response is terminated with the Untalk command, this controller may no longer be addressed to talk when the service routine begins its execution. In such a case, the SPAS interrupt (if enabled) will also be indicated. If desired, the interrupt may be ignored.

## Interface-State Information

It is often necessary to determine which state the interface is in. STATUS register 6 contains interface-state information in its upper byte; it also contains the same information as STATUS register 3 in its lower byte. In advanced applications, it may be necessary to detect and act on the interface's current state. Register 6's definition is shown below.

**STATUS Register 6**

**Interface State Information**

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
REM	LLO	ATN True	LPAS	TPAS	LADS	TADS	*
Value= -32 768	Value= 16 384	Value= 8 192	Value= 4 096	Value= 2 048	Value= 1 024	Value= 512	Value= 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Active Controller	0	Primary Address of HP-IB Interface				
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

c\* Least-significant bit of last address recognized.

**Bit 15** set indicates that the interface is in the Remote state.

**Bit 14** set indicates that the interface is in the Local Lockout state.

**Bit 13** set indicates that the ATN line is currently set (true).

**Bit 12** set indicates that the interface is in the Listener Primary Addressed State (has received its primary listen address).

**Bit 11** set indicates that the interface is in the Talker Primary Addressed State (has received its primary talk address).

**Bit 10** set indicates that the interface is in the Listener Addressed State and is currently an active listener. If Bit 4 of the Interrupt Enable register is set (Secondary Command While Addressed), two additional conditions are required to enter this state: the interface must have first received its own primary address followed by a secondary command, and it must have **accepted** the secondary command (by writing a non-zero value to CONTROL register 4 to release the NDAC Holdoff).



**Bit 9** set indicates that the interface is in the Talker Addressed State and is currently an active talker. This state is entered in a manner analogous to the Listener Addressed State (see Bit 10 above).

**Bit 8** contains the least-significant bit of the last address recognized by this interface.

**Bits 7 through 0** have the same definitions as STATUS register 3.

## Servicing Interrupts that Require Data Transfers

During the discussion on interrupts, three special types of interrupt conditions were described (which are enabled by setting bits in CONTROL register 4). These interrupts occur upon receiving: an unrecognized Universal Command, an unrecognized Addressed Command, or a Secondary Command. These situations all require the computer to read a byte of information from the bus and respond as desired by the programmer.

### STATUS Register 4

### Interrupt Status

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel Poll Configuration Change	My Talk Address Received	My Listen Address Received	EOI Received	SPAS	Remote/Local Change	Talker/Listener Address Change
Value=-32768	Value=16384	Value=8192	Value=4096	Value=2048	Value=1024	Value=512	Value=256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger Received	Handshake Error	Unrecognized Universal Command	Secondary Command While Addressed	Clear Received	Unrecognized Addressed Command	SRQ Received	IFC Received
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

As a reminder, these interrupt conditions occur under the following circumstances.

**Bit 14** enables an interrupt on any change in Parallel Poll configuration. If a Parallel Poll Configure command is received, the computer must set up its own Parallel Poll Response designated by the Active Controller. The response itself is set up by writing to CONTROL register 2 of the HP-IB interface.

**Bit 5** enables an interrupt upon receiving an unrecognized Universal Command. This interrupt condition provides the computer with the ability to respond to new definitions that may be adopted by the IEEE standards committee.

**Bit 4** enables an interrupt upon receiving a Secondary Command, if addressed to either talk or listen during the command mode. Again, this allows the computer to detect and respond to special information from another controller.

**Bit 2** enables an interrupt upon receiving an unrecognized Addressed Command, if addressed to listen. This interrupt is used to detect and respond to commands that are undefined by the standard (but which may be recognized by the computer).

Whenever any of the above interrupt conditions are enabled and occur, the computer logs the interrupt and then sets a **bus holdoff**. In other words, all bus activity is “frozen” until the program has released this holdoff. The holdoff is established to allow the program time to determine the current state of the bus.

The bus state is determined by reading HP-IB STATUS register 7, which returns the current logic state of the data and control lines as a 16-bit integer.

**STATUS 7,7;Bus\_lines**

After reading the state of the lines, it is necessary to release the bus holdoff by writing any value into HP-IB CONTROL register 4.

**CONTROL 7,4;Any\_value**

#### **CONTROL Register 4**

#### **Release NDAC Holdoff**

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
0 = Don't Accept Secondary Command All Non-zero Values Accept Secondary (Writing anything to this register releases NDAC holdoff)							
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

When a Secondary Command is received, two computer responses are possible.

- The first is to accept the address as a valid secondary address and consequently become an Extended Talker or Listener.
- The second is not to accept the address as valid and consequently remain in the primary addressed state.

If Secondary Command interrupts are enabled (while the computer is a non-Active Controller), the computer will not respond to its primary address alone; a valid secondary address is also required. Statements such as ENTER 7, OUTPUT 7, and LIST #7 should only be executed in the interrupt service routine after CONTROL has been used to indicate that a valid secondary address has been received but before interrupts are re-enabled.

When you no longer want the computer to respond as an Extended Talker/Listener, execute an ENABLE INTR with a mask which has bit 4 equal to zero.



## HP-IB Control Lines

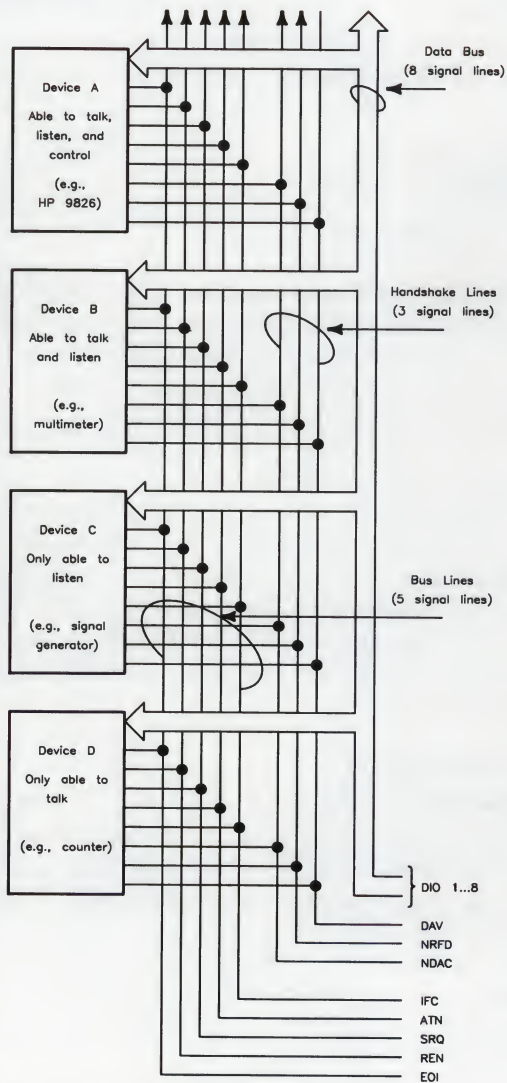


Figure 12-2. HP-IB Control Lines

## Handshake Lines

The preceding figure shows the names given to the eight control lines that make up the HP-IB. Three of these lines are designated as the “handshake” lines and are used to control the timing of data byte exchanges so that the talker does not get ahead of the listener(s). The three handshake lines are as follows.

DAV	Data Valid
NRFD	Not Ready for Data
NDAC	Not Data Accepted

The **HP-IB interlocking handshake** uses the lines as follows. All devices currently designated as active listeners would indicate when they are ready for data by using the NRFD line. A device not ready would pull this line low (true) to signal that it is not ready for data, while any device that is ready would let the line float high. Since an active low overrides a passive high, this line will stay low until all active listeners are ready for data.

When the talker senses that all devices are ready, it places the next data byte on the data lines and then pulls DAV low (true). This tells the listeners that the information on the data lines is valid and that they may read it. Each listener then accepts the data and lets the NDAC line float high (false). As with NRFD, only when all listeners have let NDAC go high will the talker sense that all listeners have read the data. It can then float DAV (let it go high) and start the entire sequence over again for the next byte of data.

## The Attention Line (ATN)

Command messages are encoded on the data lines as 7-bit ASCII characters, and are distinguished from normal data characters by the logic state of the attention line (ATN). That is, when ATN is **false**, the states of the data lines are interpreted as **data**. When ATN is **true**, the data lines are interpreted as **commands**. The set of 128 ASCII characters that can be placed on the data lines during this ATN-true mode are divided into four classes by the states of data lines DIO6 and DIO7. These classes of commands are shown in a table in the section called “Advanced Bus Management”. Only the Active Controller can set ATN true.

## **The Interface Clear Line (IFC)**

Only the System Controller can set the IFC line true. By asserting IFC, all bus activity is unconditionally terminated, the System Controller regains the capability of Active Controller (if it has been passed to another device), and any current talker and listeners become unaddressed. Normally, this line is only used to terminate all current operations, or to allow the System Controller to regain control of the bus. It overrides any other activity that is currently taking place on the bus.

## **The Remote Enable Line (REN)**

This line is used to allow instruments on the bus to be programmed remotely by the Active Controller. Any device that is addressed to listen while REN is true is placed in the Remote mode of operation.

## **The End or Identify Line (EOI)**

Normally, data messages sent over the HP-IB are sent using the standard ASCII code and are terminated by the ASCII line-feed character, CHR\$(10). However, certain devices may wish to send blocks of information that contain data bytes which have the bit pattern of the line-feed character but which are actually part of the data message. Thus, no bit pattern can be designated as a terminating character, since it could occur anywhere in the data stream. For this reason, the EOI line is used to mark the end of the data message.

The EOI line is used as an END indication (ATN false) during ENTER statements and as the Identify message (ATN true) during an identify sequence (the response to parallel poll). During data messages, the EOI line is set true by the talker to signal that the current data byte is the last one of the data transmission. Generally, when a listener detects that the EOI line is true, it assumes that the data message is concluded. However, EOI may either be used or ignored by the computer when entering data with an ENTER statement that uses an image. Chapter 5 fully describes the definitions of EOI during all ENTER statements and shows how to use the image specifiers that modify the statement-termination conditions.



ENTER statements can use images to re-define the meaning of EOI to provide a very great degree of flexibility. Using the “#” or “%” specifier in an ENTER statement affects the definition of the EOI signal as shown in the following table.

**Table 12-4. Definition of EOI During ENTER Statements**

	<b>Free-Field ENTER Statements</b>	<b>ENTER USING without # or %</b>	<b>ENTER USING with #</b>	<b>ENTER USING with %</b>
<b>Definition of EOI</b>	Immediate statement terminator	Item terminator or statement terminator	Item terminator or statement terminator	Immediate statement terminator
<b>Statement Terminator Required?</b>	Yes	Yes	No	No
<b>Early Termination Allowed?</b>	No	No	No	Yes

### **The Service Request Line (SRQ)**

The Active Controller is always in charge of the order of events that occur on the HP-IB. If a device on the bus needs the Active Controller's help, it can set the Service Request line true. This line sends a request, not a demand, and it is up to the Active Controller to choose when and how it will service that device. However, the device will continue to assert SRQ until it has been “satisfied”. Exactly what will satisfy a service request depends on the requesting device, which is explained in the device's operating manual.

## Determining Bus-Line States

STATUS register 7 contains the current states of all bus hardware lines. Reading this register returns the states of these lines in the specified numeric variable.

STATUS Hpib,7;Bus\_lines

### STATUS Register 7

### Bus Control and Data Lines

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
ATN True	DAV True	NDAC <sup>1</sup> True	NRFD <sup>1</sup> True	EOI True	SRQ <sup>2</sup> True	IFC True	REN True
Value= -32 768	Value= 16 384	Value= 8 192	Value= 4 096	Value= 2 048	Value= 1 024	Value= 512	Value= 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

### Note

Due to the way the bi-directional buffers work, NDAC and NRFD are not accurately read by this STATUS statement unless the interface is currently addressed to talk. Also, SRQ is not accurately shown unless the interface is currently the active controller.

<sup>1</sup> Only if currently Addressed to Talk, else not valid.

<sup>2</sup> Only if currently Active Controller, else not valid.

---

## Summary of HP-IB STATUS and CONTROL Registers

**Status Register 0** Card identification = 1

**Control Register 0** Reset interface if non-zero

### Status Register 1

### Interrupt and DMA Status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupts Enabled	Interrupt Requested	Hardware Level	Interrupt Switches	0	0	DMA Channel 1 Enabled	DMA Channel 0 Enabled
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

### Control Register 1

### Serial Poll Response Byte

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Device Dependent Status	SRQ 1=I did it 0=I didn't	Device Dependent Status					
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1



## HP-IB Status and Control Registers (cont.)

Status Register 2

Busy Bits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	Reserved For Future Use	Hand- shake In Progress	Interrupts Enabled	TRANS- FER In Progress
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Control Register 2

Parallel Poll Response Byte

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8 1=True	DIO7 1=True	DIO6 1=True	DIO5 1=True	DIO4 1=True	DIO3 1=True	DIO2 1=True	DIO1 1=True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Status Register 3

Controller Status and Address

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Active Controller	0	Primary Address of HP-IB Interface				
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Control Register 3

Set My Address

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used			Primary Address				
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

## HP-IB Status and Control Registers (cont.)

### Status Register 4

### Interrupt Status

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel Poll Configuration Change	My Talk Address Received	My Listen Address Received	EOI Received	SPAS	Remote/Local Change	Talker/Listener Address Change
Value=-32 768	Value=16 384	Value=8 192	Value=4 096	Value=2 048	Value=1 024	Value=512	Value=256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger Received	Handshake Error	Unrecognized Universal Command	Secondary Command While Addressed	Clear Received	Unrecognized Addressed Command	SRQ Received	IFC Received
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

### Control Register 4

Writing anything to this register releases NDAC holdoff. If non-zero, accept last secondary address as valid. If zero, don't accept last secondary address (stay in LPAS or TPAS state).

## HP-IB Status and Control Registers (cont.)

Status Register 5

Interrupt Enable Mask

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel Poll Configuration Change	My Talk Address Received	My Listen Address Received	EOI Received	SPAS	Remote/Local Change	Talker/Listener Address Change
Value=-32 768	Value=16 384	Value=8 192	Value=4 096	Value=2 048	Value=1 024	Value=512	Value=256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger Received	Handshake Error	Unrecognized Universal Command	Secondary Command While Addressed	Clear Received	Unrecognized Addressed Command	SRQ Received	IFC Received
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Control Register 5

Parallel Poll Response Mask

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used			Unconfigure	Logic Sense	Data Bit	Used for	Response
Value=128	Value=64	Value=0	Value=16	Value=8	Value=4	Value=2	Value=1



## HP-IB Status and Control Registers (cont.)

### Status Register 6

### Interface Status

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
REM	LLO	ATN True	LPAS	TPAS	LADS	TADS	*
Value= -32 768	Value= 16 384	Value= 8 192	Value= 4 096	Value= 2 048	Value= 1 024	Value= 512	Value= 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Active Controller	0	Primary Address of Interface				
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

\* Least-significant bit of last address recognized

### Status Register 7

### Bus Control and Data Lines

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
ATN True	DAV True	NDAC <sup>1</sup> True	NRFD <sup>1</sup> True	EOI True	SRQ <sup>2</sup> True	IFC True	REN True
Value= -32 768	Value= 16 384	Value= 8 192	Value= 4 096	Value= 2 048	Value= 1 024	Value= 512	Value= 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

<sup>1</sup> Only if currently Addressed to Talk, else not valid.

<sup>2</sup> Only if currently Active Controller, else not valid.

## HP-IB Status and Control Registers (cont.)

### Interrupt Enable Register (ENABLE INTR)

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel Poll Configuration Change	My Talk Address Received	My Listen Address Received	EOI Received	SPAS	Remote/Local Change	Talker/Listener Address Change
Value= -32 768	Value= 16 384	Value= 8 192	Value= 4 096	Value= 2 048	Value= 1 024	Value= 512	Value= 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger Received	Handshake Error	Unrecognized Universal Command	Secondary Command While Addressed	Clear Received	Unrecognized Addressed Command	SRQ Received	IFC Received
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

---

## Summary of HP-IB READIO and WRITEIO Registers

### READIO Registers

Register 1 — Card Identification  
Register 3 — Interrupt and DMA Status  
Register 5 — Controller Status and Address  
Register 17 — Interrupt Status 0<sup>1</sup>  
Register 19 — Interrupt Status 1<sup>1</sup>  
Register 21 — Interface Status  
Register 23 — Control-Line Status  
Register 29 — Command Pass-Through  
Register 31 — Data-Line Status<sup>1</sup>

#### HP-IB READIO Register 1

#### Card Identification

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Future Use Jumper Installed	0	0	0	0	0	0	1
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**Bit 7** is set (1) if the “future use” jumper is installed and clear (0) if not.

**Bits 6 through 0** constitute a card identification code (=1 for all HP-IB cards).

---

#### Note

This register is only implemented on external HP-IB cards. The internal HP-IB, at interface select code 7, “floats” this register (i.e., the states of all bits are indeterminate).

---

---

<sup>1</sup> Indicates that a READIO operation will change the state of the interface.



**HP-IB READIO Register 3****Interrupt and DMA Status**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupt Enabled	Interrupt Requested	Interrupt Level		X	X	DMA1	DMA0
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**Bit 7** is set (1) if interrupts are currently enabled.

**Bit 6** is set (1) when the card is currently requesting service.

**Bits 5 and 4** constitute the card's hardware interrupt level (a switch setting on all external cards, but fixed at level 3 on the internal HP-IB).

Bit 5	Bit 4	Hardware Interrupt Level
0	0	3
0	1	4
1	0	5
1	1	6

**Bits 3 and 2** are not used (indeterminate).

**Bit 1** is set (1) if DMA channel one is currently enabled.

**Bit 0** is set (1) if DMA channel zero is currently enabled.

---

**Note**

Bits 7, 5, 4, 3, 2, and 1 are not implemented on the internal HP-IB (interface select code 7).

---

## HP-IB READIO Register 5

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Not Active Controller	X	<div> <div>←</div> <div>HP-IB Primary Address of Interface</div> <div>→</div> </div> <div>(MSB) (LSB)</div>				
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**Bit 7** is set (1) if the interface is the System Controller.

**Bit 6** is set (1) if the interface is **not** the current Active Controller and clear (0) if it is the Active Controller.

**Bit 5** is not used.

**Bits 4 through 0** contain the card's Primary Address switch setting. The following bit patterns indicate the specified addresses.

Bit 4 3 2 1 0	Primary Address
0 0 0 0 0	0
0 0 0 0 1	1
:	:
:	:
1 1 1 0 1	29
1 1 1 1 0	30
1 1 1 1 1	(not allowed)

---

### Note

Bits 5 through 0 are not implemented on the internal HP-IB.

---

**HP-IB READIO Register 17****MSB of Interrupt Status**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MSB Interrupt	LSB Interrupt	Byte Received	Ready for Next Byte	End Detected	SPAS	Remote/ Local Change	My Address Change
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**Bit 7** set (1) indicates that an interrupt has occurred whose cause can be determined by reading the contents of this register.

**Bit 6** set (1) indicates that an interrupt has occurred whose cause can be determined by reading Interrupt Status Register 1 (READIO Register 19).

**Bit 5** set (1) indicates that a data byte has been received.

**Bit 4** set (1) indicates that this interface is ready to accept the next data byte.

**Bit 3** set (1) indicates that an End (EOI with ATN=0) has been detected.

**Bit 2** set (1) indicates that a Remove/Local State change has occurred.

**Bit 0** set (1) indicates that a change in My Address has occurred.



**HP-IB READIO Register 19****LSB of Interrupt Status**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger Received	Hand- shake Error	Unrecog- nized Command Group	Secondary Command While Addressed	Clear Received	My Address Received (MLA or MTA)	SRQ Received	IFC Received
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**Bit 7** set (1) indicates that a Group Execute Trigger command has been received.

**Bit 6** set (1) indicates that an Incomplete-Source-Handshake error has occurred.

**Bit 5** set (1) indicates that an unidentified command has been received.

**Bit 4** set (1) indicates that a Secondary Address has been sent in while in the extended-addressing mode.

**Bit 3** set (1) indicates that the interface has entered the Device-Clear-Active State.

**Bit 2** set (1) indicates that My Address has been received.

**Bit 1** set (1) indicates that a Service Request has been received.

**Bit 0** set (1) indicates that the Interface Clear message has been received.

**HP-IB READIO Register 21****Interface Status**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
REM	LLO	ATN True	LPAS	TPAS	LADS	TADS	LSB of Last Address
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**Bit 7** set (1) indicates that this interface is in the Remote State.

**Bit 6** set (1) indicates that this interface is in the Local Lockout State.

**Bit 5** set (1) indicates that the ATN signal line is true.

**Bit 4** set (1) indicates that this interface is in the Listener-Primary-Addressed State.

**Bit 3** set (1) indicates that this interface is in the Talker-Primary-Addressed State.

**Bit 2** set (1) indicates that this interface is in the Listener-Addressed State.

**Bit 1** set (1) indicates that this interface is in the Talker-Addressed State.

**Bit 0** set (1) indicates that this is the least-significant bit of the last address recognized by this interface.

**HP-IB READIO Register 23****Control-Line Status**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ATN True	DAV True	NDAC <sup>1</sup> True	NRFD <sup>1</sup> True	EOI True	1SRQ <sup>2</sup> True	IFC True	REN True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

A set bit (1) indicates that the corresponding line is currently true; a 0 indicates that the line is currently false.

**HP-IB READIO Register 29****Command Pass-Through**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

This register can be read during a bus holdoff to determine which Secondary Command has been detected.

**HP-IB READIO Register 31****Bus Data Lines**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

A set bit (1) indicates that the corresponding HP-IB data line is currently true; a 0 indicates the line is currently false.

<sup>1</sup> Only if addressed to TALK, else not valid.

<sup>2</sup> Only if Active Controller, else not valid.



## HP-IB WRITEIO Registers

Register 3 — Interrupt Enable  
Register 17 — MSB of Interrupt Mask  
Register 19 — LSB of Interrupt Mask  
Register 23 — Auxiliary Command Register  
Register 25 — Address Register  
Register 27 — Serial Poll Response  
Register 29 — Parallel Poll Response  
Register 31 — Data Out Register

### HP-IB WRITEIO Register 3

### Interrupt and DMA Enable

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Enable Interrupt	X	X	X	X	X	Enable Channel 1	Enable Channel 0
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**Bit 7** enables interrupts from this interface if set (1) and disables interrupts if clear (0).

**Bits 6 through 2** are “don’t cares” (i.e., their values have no effect on the interface’s operation).

**Bit 1** enables DMA channel 1 if set (1) and disables if clear (0).

**Bit 0** enables DMA channel 0 if set (1) and disables if clear (0).

---

#### Note

Bits 7 through 1 are not implemented on the internal HP-IB interface and thus have no effect on the interface’s operation.

---

### WRITEIO Register 17

### MSB of Interrupt Mask

Setting a bit of this register enables an interrupt for the specified condition. The bit assignments are the same as for the MSB of Interrupt Status Register (READIO Register 17), except that bits 7 and 6 are not used.

**WRITEIO Register 19****LSB of Interrupt Mask**

Setting a bit of this register enables an interrupt for the specified condition. The bit assignments are the same as for the LSB of Interrupt Status Register (READIO Register 19).

**HP-IB WRITEIO Register 23****Auxiliary Command Register**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Set	X	X	Auxiliary Command Function				
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**Bit 7** is set (1) for a Set operation and clear (0) for a Clear operation.

**Bits 6 and 5** are “don’t cares.”

**Bits 4 through 0** are Auxiliary-Command-Function-Select bits. The following commands can be sent to the interface by sending the specified numeric values.

**Table 12-5. Auxiliary Commands**

Decimal Value	Description of Auxiliary Command
0	Clear Chip Reset
128	Set Chip Reset
1	Release ACDS holdoff. If Address Pass Through is set, it indicates an invalid secondary has been received.
129	Release ACDS holdoff. If Address Pass Through is set, indicates a valid secondary has been received.
2	Release RFD holdoff.
130	Same command as decimal 2 (above).
3	Clear holdoff on all data.
131	Set holdoff on all data.
4	Clear holdoff on EOI only.
132	Set holdoff on EOI only.
5	Set New Byte Available (nba) false.
133	Same command as decimal 5 (above).
6	Pulse the Group Execute Trigger line, or clear the line if it was set by decimal command 134.
134	Set Group Execute Trigger line.
7	Clear Return To Local (rtl).
135	Set Return To Local (must be cleared before the device is able to enter the Remote state).
8	Causes EOI to be sent with the next data byte.
136	Same command as decimal 8 (above).
9	Clear Listener State (also cleared by decimal 138).
137	Set Listener State.
10	Clear Talker State (also cleared by decimal 137).
138	Set Talker State.



**Table 12-5. Auxiliary Commands (continued)**

Decimal Value	Description of Auxiliary Command
11	Go To Standby (gts; controller sets ATN false).
139	Same command as decimal 11 (above).
12	Take Control Asynchronously (tca; ATN true).
140	Same command as decimal 12 (above).
13	Take Control Synchronously (tcs; ATN true).
141	Same command as decimal 13 (above).
14	Clear Parallel Poll
142	Set Parallel Poll (read Command-Pass-Through register before clearing).
15	Clear the Interface Clear line (IFC).
143	Set Interface Clear (IFC maintained > 100 $\mu$ s).
16	Clear the Remote Enable (REN) line.
144	Set Remote Enable.
17	Request control (after TCT is decoded, issue this to wait for ATN to drop and receive control).
145	Same command as decimal 17 (above).
18	Release control (issued after sending TCT to complete a Pass Control and set ATN false).
146	Same command as decimal 18 (above).
19	Enable all interrupts.
147	Disable all interrupts.
20	Pass Through next Secondary Command.
148	Same command as decimal 20 (above).
21	Set TI delay to 10 clock cycles (2 $\mu$ s at 5 MHz).
149	Set TI delay to 6 clock cycles (1.2 $\mu$ s at 5 MHz).
22	Clear Shadow Handshake
150	Set Shadow Handshake.

**HP-IB WRITEIO Register 25****Address Register**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Enable Dual Addressing	Disable Listen	Disable Talker	Primary Address				
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**Bit 7** set (1) enables the Dual-Primary-Addressing Mode.

**Bit 6** set (1) invokes the Disable-Listen function.

**Bit 5** set (1) invokes the Disable-Talker function.

**Bits 4 through 0** set the device's Primary Address (same address bit definitions as READIO Register 5).

**HP-IB WRITEIO Register 27****Serial Poll Response Byte**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Device Dependent Status	Request Service	Device Dependent Status					
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**Bits 7 and 5—0** specify the Device-Dependent Status.

**Bit 6** sends an SRQ if set (1).

---

**Note**

Given an unknown state of the Serial Poll Response Byte, it is necessary to write the byte with bit 6 set to zero followed by a write of the byte with bit 6 set to the desired final value. This will insure that a SRQ will be generated if one was desired.

---

**HP-IB WRITEIO Register 29****Parallel Poll Response**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

A 1 sets the appropriate bit true during a Parallel Poll; a 0 sets the corresponding bit false. Initially, and when Parallel Poll is not configured, this register must be set to all zeros.

**HP-IB WRITEIO Register 31****Data-Out Register**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1



## Summary of Bus Sequences

The following tables show the bus activity invoked by executing HP-IB statements and functions. The mnemonics used in these tables were defined in the previous sections of this chapter.

Note that bus messages are sent by using single lines (such as the ATN line) and multiple lines (such as DCL). The information shows the state of and changes in the state of the ATN line during these bus sequences. The tables implicitly show that **these changes in the state of ATN remain in effect unless another change is explicitly shown in the table.** For example, if a statement sets ATN (true) with a particular command, then it remains true unless the table explicitly shows that it is set false. The ATN line is implemented in this manner to avoid unnecessary transitions in this signal whenever possible. It should not cause any dilemmas in most cases.

### ABORT

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	IFC (duration $\geq 100\mu\text{sec}$ ) REN ATN	Error	ATN MTA UNL ATN	Error
Not Active Controller	IFC (duration $\geq 100\mu\text{sec}$ )* REN ATN		No Action	

\*The IFC message allows a non-active controller (which is the system controller) to become the active controller.

## CLEAR

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN DCL	ATN MTA UNL LAG SDC	ATN DCL	ATN MTA UNL LAG SDC
Not Active Controller	Error			

## LOCAL

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	<u>REN</u> ATN	ATN MTA UNL LAG GTL	ATN GTL	ATN MTA UNL LAG GTL
Not Active Controller	<u>REN</u>	Error	Error	

## LOCAL LOCKOUT

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN LLO	Error	ATN LLO	Error
Not Active Controller	Error			

## PASS CONTROL

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	Error	ATN TAD TCT ATN	Error	ATN TAD TCT ATN
Not Active Controller	Error			

## PPOLL

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN & EOI (duration $\geq 25\mu\text{s}$ ) Read byte EOI Restore ATN to previous state	Error	ATN & EOI (duration $\geq 25\mu\text{s}$ ) Read byte EOI Restore ATN to previous state	Error
Not Active Controller	Error			

## PPOLL CONFIGURE

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	Error	ATN MTA UNL LAG PPC PPE	Error	ATN MTA UNL LAG PPC PPE
Not Active Controller	Error			



## PPOLL UNCONFIGURE

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN PPU	ATN MTA UNL LAG PPC PPD	ATN PPU	ATN MTA UNL LAG PPC PPD
Not Active Controller	Error			

## REMOTE

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	REN ATN	REN ATN MTA UNL LAG	Error	
Not Active Controller	REN	Error	Error	

## SOLL

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	Error	ATN UNL MLA TAD SPE ATN Read data ATN SPD UNT	Error	ATN UNL MLA TAD SPE ATN Read data ATN SPD UNT
Not Active Controller	Error			

## TRIGGER

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN GET	ATN UNL LAG GET	ATN GET	ATN MTA UNL LAG GET
Not Active Controller	Error			





# Table of Contents

---

## Chapter 13: RS-232C Serial Interfaces

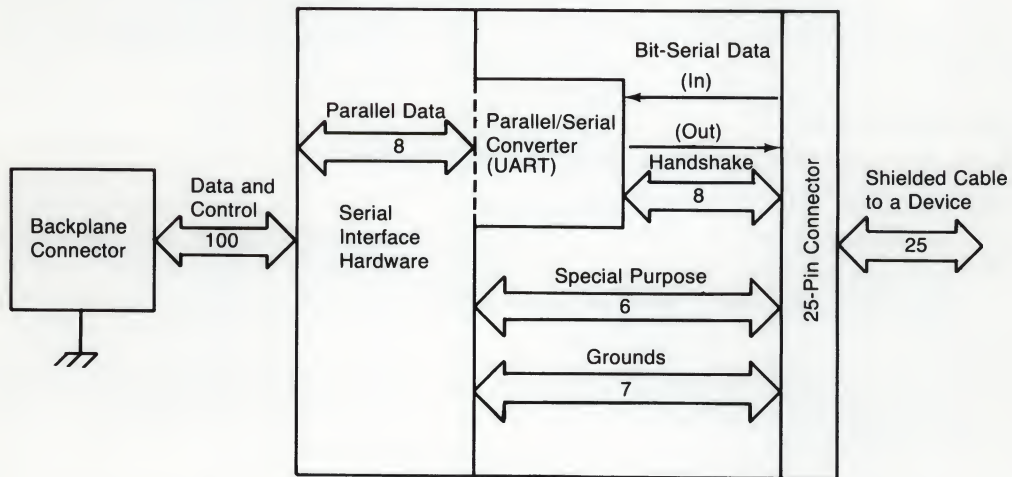
Asynchronous Data Communication .....	13-2
Data Transfers Between Computer and Peripheral .....	13-5
Overview of Serial Interface Programming .....	13-6
Determining Operating Parameters .....	13-6
Serial Configuration for BASIC/UX .....	13-7
Using Program Control to Override Defaults .....	13-9
Transferring Data .....	13-12
Entering and Outputting Data .....	13-12
Modem Line Handshaking .....	13-13
Incoming Data Error Detection and Handling .....	13-14
Advanced Programming Information .....	13-16
Sending BREAK Messages .....	13-16
Using the Modem Control Register .....	13-16
Cable Options and Signal Functions .....	13-18
The DTE Cable .....	13-19
The DCE Cable .....	13-20
RS-232C / CCITT V24 .....	13-23
HP 98626 and HP 98644 Serial Interface STATUS and CONTROL Registers	13-25
HP 98644 Interface Differences .....	13-31
Hardware Differences .....	13-31
BASIC Differences .....	13-34
Series 300 Built-In 98644 Interface Differences .....	13-35



# RS-232C Serial Interfaces

# 13

The HP 98626 and HP 98644 Serial Interfaces are RS-232C compatible interfaces used for **simple asynchronous** I/O applications such as driving line printers, terminals, or other peripherals where the more sophisticated capabilities of the HP 98628 and HP 98642 Data Communications Interfaces<sup>1</sup> are not justified. Because the HP 98626 and HP 98644 Serial Interface cards have only a few differences, this chapter will deal mainly with the HP 98626 interface card. Information on differences between these two serial interface cards can be found in the sections "HP 98644 Interface Differences" and "Series 300 Built-In 98644 Interface Differences."



**Figure 13-1. Block Diagram of the Serial Interface**

The BASIC system must provide most control functions because these cards do **not** have their own microprocessor (as do the HP 98628 and HP 98642 cards). Consequently, there is more interaction between these cards and computer than when you use a more intelligent interface except for relatively simple applications.

<sup>1</sup> See the "Datacomm Interfaces" chapter for details.



The RS-232C interface standard<sup>1</sup> establishes electrical and mechanical interface requirements, but does not define the exact function of all the signals that are used by various manufacturers of data communications equipment and serial I/O devices. Consequently, when you plug your serial interface into an RS-232 connector, there is no guarantee the devices can communicate unless you have configured optional parameters to match the requirements of the device you are connecting to.

## Asynchronous Data Communication

The terms Asynchronous (Async for short) data communication and Serial I/O refer to a technique of transferring information between two communicating devices by means of bit-serial data transmission. This means that data is sent, one bit at a time, and that characters are not synchronized with preceding or subsequent data characters; that is, each character is sent as a complete entity without relationship to other events, before or after. Characters may be sent in close succession, or they may be sent sporadically as data becomes available. Start and stop bits are used to identify the beginning and end of each character, with the character data placed between them.

### Character Format

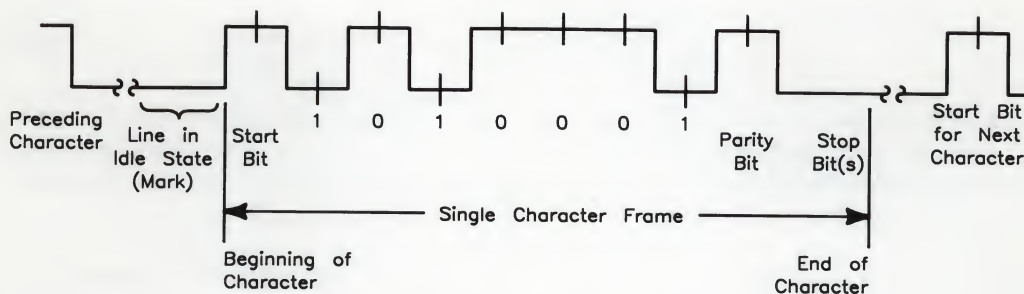
Each character frame consists of the following elements:

- **Start Bit:** The start bit signals the receiver that a new character is being sent. Since the receiver knows how many bits per second are being transmitted (specified by the baud rate), it can determine the expected arrival time for all subsequent bits in that character frame. All other bits in a given frame are synchronized to the start bit.
- **5-8 Character Data Bits:** The next bits are the binary code of the character being transmitted, consisting of 5, 6, 7, or 8 bits; depending on the application. The parity bit is not included in the character data bits.
- **Parity Bit:** The parity bit is optional, included only when parity is enabled.
- **Stop Bit(s):** One or more stop bits identify the end of each character. The serial interface has no provision for inserting time gaps between characters.

---

<sup>1</sup> RS-232C is a data communication standard established and published by the Electronic Industries Association (EIA). Copies of the standard are available from the association at 2001 Eye Street N. W., Washington D. C. 20006. Its equivalent for European applications is CCITT V.24.

Here is a simple diagram showing the structure of an asynchronous character and its relationship to other characters in the data stream:



**Figure 13-2. Asynchronous Format**

### Parity

The parity bit is used to detect errors as incoming characters are received. If the parity bit does not match the expected sense, the character is assumed to be incorrectly received. The action taken when an error is detected depends upon how the interface and the BASIC program are configured.

Parity sense is determined by system requirements. The parity bit may be included or omitted from each character by enabling or disabling the parity function. If the parity bit is enabled, four options are available. Parity is checked by the receiver for all parity options including ONE and ZERO. (The HP 98628 and HP 98642 Datacomm Interface cards do not check parity when parity is set to ONE or ZERO.)

Parity options include:

- NONE Parity function is DISABLED, and the parity bit is omitted from each character frame.
- ODD Parity bit is SET if there is an EVEN<sup>1</sup> number of ones in the data character. The receiver performs parity checks on incoming characters.
- EVEN Parity bit is SET if there is an ODD<sup>1</sup> number of ones in the data character. The receiver performs parity checks on incoming characters.

<sup>1</sup> Parity sense is determined by counting the number of ones in the character *including* the parity bit. Consequently, the parity sense is reversed from the number of ones in a character without the parity bit.

## Error Detection

Two types of incoming data errors can be detected by serial receivers:

- **Parity errors** are signaled when the parity bit does not match the number of ones, including the parity bit, even or odd as defined by interface configuration. When parity is disabled, no parity check is made.
- **Framing errors** are signaled when start and stop bits are not properly received during the expected time frame. They can be caused by a missing start bit, noise errors near the end of the character, or by improperly specified character length at the transmitter or receiver.

Two additional error types are detected by the receiver section of the serial interface:

- **Overrun errors** result when the desktop computer does not consume characters as fast as they arrive. The card provides only one character of buffer space, so the current character must be consumed by an ENTER before the next character arrives. Otherwise, the character is lost when the next character replaces it, and an error is sent to BASIC.
- **Received BREAKs** are detected as a special type of framing error. They generate the same type of BASIC error as framing errors.



## Data Transfers Between Computer and Peripheral

Five statements are used to transfer information between your desktop computer and the interface card:

- The OUTPUT statement sends data to the interface which, in turn, sends the information to the peripheral device.
- The ENTER statement inputs data from the interface card after the interface has received it from the peripheral device.
- The STATUS statement is used to monitor the interface and obtain information about interface operation such as buffer status, detected errors, and interrupt enable status.
- The CONTROL statement is used to control interface operation and defines such parameters as baud rate, character format, or parity.
- The TRANSFER statement is used to input or output data from/to the interface and, in turn, from/to the peripheral device.

Since the interface has no on-board processor, ENTER and OUTPUT statements cause the computer to wait until the ENTER or OUTPUT operation is complete before continuing to the next line. For OUTPUT statements, this means that the computer waits until the last bit of the last character has been sent over the serial line before continuing with the next program statement.

---

## Overview of Serial Interface Programming

Serial interface programming techniques are similar to most general I/O applications. The interface card is initialized by use of CONTROL statements; STATUS statements evaluate its readiness for use. Data is transferred between the desktop computer and a peripheral device by OUTPUT and ENTER statements.

Due to the asynchronous nature of serial I/O operations, special care must be exercised to ensure that data is not lost by sending to another device before the device is ready to receive. Modem line handshaking can be used to help solve this problem. These and other topics are discussed in greater detail elsewhere in this chapter.

### Determining Operating Parameters

Before you can successfully transfer information to a device, you must match the operating characteristics of the interface to the corresponding characteristics of the peripheral device. This includes matching signal lines and their functions as well as matching the character format for both devices.

#### Handshake and Baud Rate

To determine hardware operating parameters, you need to know the answer for each of the following questions about the peripheral device:

- Which of the following signal and control lines are actively used during communication with the peripheral?

Data Set Ready (DSR)

Data Carrier Detect (DCD or CD)

Clear to Send (CTS)

Ring Indicator (RI)

- What baud rate (line speed) is expected by the peripheral?

## Character Format Parameters

To define the character format, you must know the requirements of the peripheral device for the following parameters:

- **Character Length:** How many data bits are used for each character, excluding start, stop, and parity bits?
- **Parity Enable:** Is parity enabled (included) or disabled (absent) for each character?
- **Parity Sense:** Is the parity bit, if enabled, ODD, EVEN, always ONE, or always ZERO?
- **Stop Bits:** How many stop bits are included with each character: 1, 1.5, or 2?

## Serial Configuration for BASIC/UX

There is no capability in BASIC/UX for reading the hardware bit settings on either the HP 98626 or HP 98644 Serial Interface cards. Therefore, BASIC/UX provides two methods for configuring modem control options:

- The **stty** command from the HP-UX environment.
- The keyword **CONTROL** and registers directly related to the modem control options.

Of the two methods mentioned above the best one to use is the **stty** command while in the HP-UX environment. The reason for this is any modem control options set by using the keyword **CONTROL** are lost when you leave BASIC/UX. However, if you prefer to change these options while in the BASIC/UX environment, then read the subsequent section "Using Program Control to Override Defaults."

This section deals with the first method mentioned above which is the use of the **stty** command from the HP-UX environment.

### Defaults for the Serial Interface

When HP-UX is being booted up, the defaults for all serial interfaces are:

Baud rate	300
Bits per character	7
Parity	Odd
Stop bits	2

The above values are used by BASIC/UX as defaults, unless configured as explained in the next section.



Some common serial interface configuration settings are:

Baud rate to	9600
Bits per character to	8
Parity to	Odd and disabled
Stop bits to	1

### Configuring a Serial Interface for BASIC/UX

To configure your serial interface with the values mentioned in the previous section, you can execute the following HP-UX command before entering BASIC/UX:

```
/bin/stty 9600 cs8 -parenb parodd -cstopb < /dev/rmb/serialnn
```

where:

**9600** is the baud rate. The following are baud rates you can use with the *stty* command:

50	150	1200	4800
75	200	1800	7200
110	300	2400	9600
134	600	3600	19200

**cs8** is the number of bits per character. In the case of this example, the number of bits per character is 8. Other character lengths can be set using **cs5**, **cs6**, or **cs7** for 5, 6, or 7 bits per character respectively.

**-parenb** disables parity generation and detection. Removing the minus sign that is prefixed to this *stty* option causes parity generation and detection to be enabled.

**parodd** selects odd parity. Prefixing a minus sign to this *stty* option selects even parity.

- `-cstopb` causes one stop bit per character to be used. Removing the minus sign that is prefixed to this `stty` option causes two stop bits per character to be used.
- `< /dev/rmb/serialnn` assigns the `stty` options to the serial interface located at select code number *nn*.

For more information on `stty` options, see the *HP-UX Language Reference*.

## Using Program Control to Override Defaults

You can override some of the interface default configuration options by use of `CONTROL` statements. This not only enables you to guarantee certain parameters, but also provides a means for changing selected parameters in the course of a running program. `CONTROL` Register tables are listed at the end of this chapter as well as in the *BASIC Language Reference*. Refer to them as needed during the discussion which follows.

### Interface Reset

Whenever an interface is connected to a modem that may still be connected to a telecommunications link from a previous session, it is good programming practice to reset the interface to force the modem to disconnect, unless the status of the link and remote connection are known. When the interface is connected to a line printer or similar peripheral, resetting the interface is usually unnecessary unless an error condition requires it.

```
100 CONTROL Sc,0;1 ! Reset interface.
```

When the interface is reset by use of a `CONTROL` statement to `CONTROL` Register 0 with a non-zero value, the interface is restored to the BASIC/UX power-up condition whether or not it is the same as the current default switch configuration. If you are not sure of the present settings, or if your application requires changing the configuration during program operation, you can use `CONTROL` statements to configure the interface. An example of where this may be necessary is when several peripherals share a single interface through a manually operated RS-232 switch such as those used to connect multiple terminals to a single computer port, or a single terminal to multiple computers.

### Selecting the Baud Rate

In order to successfully transfer information between the interface card and a peripheral, the interface and peripheral must be set to the same baud rate. A CONTROL statement to register 3 (or 13 with 98644 interfaces) can be used to set the interface baud rate to any one of the following values:

50	150	1200	4800
75	200	1800	7200
110	300	2400	9600
134	600	3600	19 200

For example, to select a baud rate of 3600, the following program statement is used:

```
1190 CONTROL Sc,3;3600
```

Use of values other than those shown may result in incorrect operation.

To verify the current baud rate setting, use a STATUS statement addressed to register 3. All rates are in baud (bits/second).



### Setting Character Format and Parity

CONTROL Register 4 overrides the Line Control switches<sup>1</sup> that control parity and character format. To determine the value sent to the register, add the appropriate values selected from the following table:

**Table 13-3. Character Format and Parity Settings**

Handshake (Bits <sup>3</sup> 7&6)	Parity Sense <sup>2</sup> (Bits <sup>3</sup> 5&4)	Par. Enable (Bit <sup>3</sup> 3)	Stop Bits (Bit <sup>3</sup> 2)	Char. Length (Bits <sup>3</sup> 1&0)
00 no-op	00 ODD parity	0 Disabled	0 1 stop bit	00 5 bits/char
01 Xon/Xoff	01 EVEN parity	1 Enabled	1 2 stop bits	01 6 bits/char
Bidirectional	10 Unsupported			10 7 bits/char
10 Unsupported	11 Unsupported			11 8 bits/char
11 Handshake Disabled				

For example, to configure a character format of 8 bits per character, two stop bits, and EVEN parity, use the following CONTROL statement:

```
1200 CONTROL Sc,4;IVAL("11111",2)
```

or

```
1200 CONTROL Sc,4;31
```

To configure a 5-bit character length with 1 stop bit and no parity bit, use the following:

```
1200 CONTROL Sc,4;0
```

<sup>1</sup> With 98644 interfaces, there are no Line Control switches. You can simulate their effect by writing to CONTROL register 14. Note that individual bits of this register are the same as for register 4.

<sup>2</sup> Parity sense valid only if parity is enabled (bit 3=1). If parity is disabled, parity sense is meaningless.

<sup>3</sup> These bits correspond to equivalent switch settings on the HP 98626 and HP 98644 serial interface cards. A 1 is the same as set.

---

## Transferring Data

The serial interface card is designed for relatively simple serial I/O operations. It is not intended for sophisticated applications that use ON INTR statements to service the interface.

### Entering and Outputting Data

When the interface is properly configured, either by use of default switches or CONTROL statements, you are ready to begin data transfers. OUTPUT statements are used to send information to the peripheral; ENTER statements to input information from the external device.

```
OUTPUT 20;"String data",Numeric_var,Etc
```

```
ENTER 20;String_var$,Numeric_var,Etc
```

Any valid OUTPUT or ENTER statement and variable(s) list may be used, but you must be sure that the data format is compatible with the peripheral device. For example, non-ASCII data sent to an ASCII line printer may result in unexpected behavior.

Various other I/O statements can be used in addition to OUTPUT and ENTER, depending on the situation. For example, the LIST statement can be used to list programs to an RS-232 line printer—provided the interface is properly configured *before* the operation begins.

### Outputting Data

To send data to a peripheral, use OUTPUT, OUTPUT USING, or any other similar or equivalent construct. Suppression of end-of-line delimiters and other formatting capabilities are identical to normal operation in general I/O applications. The OUTPUT statement hangs the computer until the last bit of the last character in the statement variable list is transmitted by the interface. When the output operation is complete, the computer then continues to the next line in the program. See the "Outputting Data" chapter for details of the OUTPUT statement.

## Entering Data

To input data from a peripheral, use ENTER, ENTER USING, or an equivalent statement. Inclusion or elimination of end-of-line delimiters and other information is determined by the formatting specified in the ENTER statement. The ENTER statement hangs the computer until the input variables list is satisfied. To minimize the risk of waiting for another variable that isn't coming, you may prefer to specify only one variable for each ENTER statement, and analyze the result before starting the next input operation. See the "Entering Data" chapter for details of the ENTER statement.

Be sure that the peripheral is not transmitting data to the interface while no ENTER is in progress. Otherwise, data may be lost because the card provides buffering for only one character. Also, interrupts from other I/O devices, or operator inputs to the computer keyboard can cause delays in computer service to the interface that result in buffer overrun at higher baud rates.

## Modem Line Handshaking

Modem line handshaking, when used, is performed automatically by the computer as part of the OUTPUT or ENTER operation. If the modem line states have not been latched in a fixed state by Control Register 5, the following sequence of events is executed automatically during each OUTPUT or ENTER operation:

For OUTPUT operations:

1. Set Data Terminal Ready and Request-to-Send modem lines to active state.
2. Check Data Set Ready and Clear-to-Send modem lines to be sure they are active.
3. Send information to the interface and thence to the peripheral.
4. After data transfer is complete, clear Data Terminal Ready and Request-to-Send signals.

For ENTER operations:

1. Set Data Terminal Ready line to active state. Leave Request-to-Send inactive.
2. Check Data Set Ready and Data Carrier Detect modem lines to be sure they are active.
3. Input information from the interface as it is received from the peripheral.
4. After the input operation is complete, clear the Data Terminal Ready signal.



After a given OUTPUT or ENTER operation is completed, the program continues execution on the next line.

Control Register 5 can be used to force selected modem control lines to their active state(s). The Data Rate Select line is set or cleared by bit 2. Request-to-send and Data Terminal Ready are held in their active states when bits 1 and 0 are true, respectively. If bits 1 and/or 0 are false, the corresponding modem line is toggled during OUTPUT or ENTER as explained previously.

### **Incoming Data Error Detection and Handling**

The serial interface card can generate several errors that are caused when certain conditions are encountered while receiving data from the peripheral device. The UART detects a given error condition. The card then generates a pending error to BASIC. Errors can be generated by any of the following conditions:

- **Parity error.** The parity bit on an incoming character does not match the parity expected by the receiver. This condition is most commonly caused by line noise.
- **Framing error.** Start and stop bit(s) do not match the timing expectations of the receiver. This can occur when line noise causes the receiver to miss the start bit or obscures the stop bits.
- **Overrun error.** Incoming data buffer overrun caused a loss of one or more data characters. This is usually caused when data is received by the interface, but no ENTER statement has been activated to input the information.
- **Break received.** A BREAK was sent to the interface by the peripheral device. The desktop computer program must be able to properly interpret the meaning of a break and take appropriate action.

All UART status errors are generated by *incoming* data, never by outbound data. When a UART error occurs, the corresponding bit of Status Register 10 is set, and a pending error (ERROR 167: Interface status error) is sent to BASIC. BASIC processes the error according to the following rules:

- If an ENTER is in progress, the error is handled immediately as part of the ENTER process. An active ON ERROR causes the error trap to be executed. If no ON ERROR is active, the error is fatal and causes the program to terminate.
- If an OUTPUT is in progress, or if there is no current activity between the computer and interface, the error is flagged, but nothing is done by BASIC until an ENTER statement is encountered. When the computer begins execution of the ENTER statement, if an ON ERROR is active, the error trap is executed. If there is no active ON ERROR for that select code, the fatal ERROR 167 causes the BASIC program to terminate.
- If a STATUS statement is executed to Status Register 10 before an ENTER statement is encountered for that select code, the pending BASIC error is cleared, and the program continues as if no error had been generated.

---

## Advanced Programming Information

This section provides advanced programming information for applications requiring special techniques.

### Sending BREAK Messages

A BREAK is a special character transmission that usually indicates a change in operating conditions. Interpretation of break messages varies with the application. To send a break message, send a non-zero value to Control Register 1 as follows (Sc is the interface select code):

```
1640 CONTROL Sc,1;1      ! Send a BREAK to peripheral.
```

### Using the Modem Control Register

Control Register 5 controls various functions related to modem operation. Bits 0 thru 3 control modem lines, and bit 4 enables a self-test loopback configuration.

#### Modem Handshake Lines (RTS and DTR)

As explained earlier in this chapter, Request-to-send and Data Terminal Ready lines are set or cleared at the beginning and end of each OUTPUT or ENTER operation. In some cases, it may be advantageous or necessary to maintain either or both in an active state. This is done by setting bit 1 or 0 respectively in Control Register 5 as follows:

```
1650 CONTROL Sc,5;2      ! Set RTS line only and hold active.
1660 CONTROL Sc,5;1      ! Set DTR line only and hold active.
1670 CONTROL Sc,5;3      ! Set both RTS and DTR lines active.
1680 CONTROL Sc,5;0      ! Return to normal modem line handshake.
```

When RTS and/or DTR are set by Control Register 5, they are *not* toggled during OUTPUT or ENTER operations, but remain constantly in an active state until the CONTROL register is cleared by:

- writing a different value to CONTROL register 5
- an interface reset to CONTROL register 0
- an interface reset (**Reset**) from the keyboard (**Shift** **Break** on an ITF keyboard, or **SHIFT**-**PAUSE** on a 98203 keyboard).



### **Programming the DRS Modem Line**

Bit 2 of Control Register 5 controls the present state of the Data Rate Select (DRS). When bit 2 is set, the modem line is activated. When bit 2 is cleared, the modem line is cleared. To set the DRS line, the following statement or its equivalent can be used:

```
1690 CONTROL Sc,5;4    ! Sets the DRS line.
```

This line is also cleared by a CONTROL statement to Control Register 5 with bit 2 cleared, or by an interface reset.

---

## Cable Options and Signal Functions

The HP 98626A Serial Interface is available with RS-232C DTE and DCE cable configurations. The DTE cable option consists of a male RS-232C connector and cable designed to function as Data Terminal Equipment (DTE) when used with the serial interface. This means that the cable and connector are wired so that signal paths are correctly routed when the cable is connected to a peripheral device wired as Data Communication Equipment (DCE), such as a modem. The cables are designed so that you can write programs that work for both DCE and DTE connections without requiring modifications to accommodate equipment changes.

The DCE cable option includes a female connector and cable wired so that the interface and cable behave like normal DCE. This means that signals are routed correctly when the female cable connector is connected to a male DTE connector.

Line printers and other peripheral devices that use RS-232C interfacing are frequently wired as DTE with a female RS-232C chassis connector. This means that if you use a male (DTE) cable option to connect to the female DTE device connector, no communication can take place because the signal paths are incompatible. To eliminate the problem, use an adapter cable to convert the female RS-232C chassis connector to a cable connector that is compatible with the male or female interface cable connector. The HP 13242 adapter cable is available in various configurations to fit most common applications. Consult cable documentation to determine which adapter cable to use.

## The DTE Cable

The signals and functions supported by the DTE cable are shown in the signal identification table which follows. The table includes RS-232C signal identification codes, CCITT V.24 equivalents, the pin number on the interface card rear panel connector, the RS-232C connector pin number, the signal mnemonic used in this manual, whether the signal is an input or output signal, and its function.

**Table 13-6. RS-232C DTE (Male) Cable Signal Identification Table**

RS-232C Signal	V.24 Signal	Interface Pin#	RS-232C Pin#	Mnemonic	I/O	Function
AA	101	24	1	—	—	Safety Ground
BA	103	12	2	Out		Transmitted Data
BB	104	42	3	In		Received Data
CA	105	13	4	RTS	Out	Request to Send
CB	108	44	5	CTS	In	Clear to Send
CC	107	45	6	DSR	In	Data Set Ready
AB	102	48	7	—	—	Signal Ground
CF	109	46	8	DCD	In	Data Carrier Detect
SCF (OCR2)	122	47	12	SDCD	In	Secondary DCD
SCA (OCD2)	120	15	19	SRTS	Out	Secondary RTS
CD	108.1	14	20	DTR	Out	Data Terminal Ready
CE (OCR1)	125	9	22	RI	In	Ring Indicator
CH (OCD1)	111	40	23	DRS	Out	Data Rate Select



### Optional Circuit Driver/Receiver Functions

Not all signals from the interface card are included in the cable wiring. RS-232C provides for four optional circuit drivers and two receivers. Only two drivers and two receivers are supported by the DCE and DTE cable options. They are as follows:

Drivers		Receivers	
Name	Function	Name	Function
OCD1	Data Rate Select	OCR1	Ring Indicator
OCD2	Secondary Request-to-send	OCR2	Secondary Data Carrier Detect
OCD3	Not used		
OCD4	Not used		

If your application requires use of OCD3 or OCD4, you must provide your own interface cable to fit the situation.

### The DCE Cable

The DCE cable option is designed to adapt a DTE cable and serial or data communications interface to an identical interface on another desktop computer. It is also used with the serial interface to simulate DCE operation when driving a peripheral wired for DTE operation. The DCE cable is equipped with a female connector. Since most DTE peripherals are also equipped with female connectors (pin numbering is the same as the standard male DTE connector), an adapter (such as the HP 13242M) is used to connect the two female connectors as explained earlier.

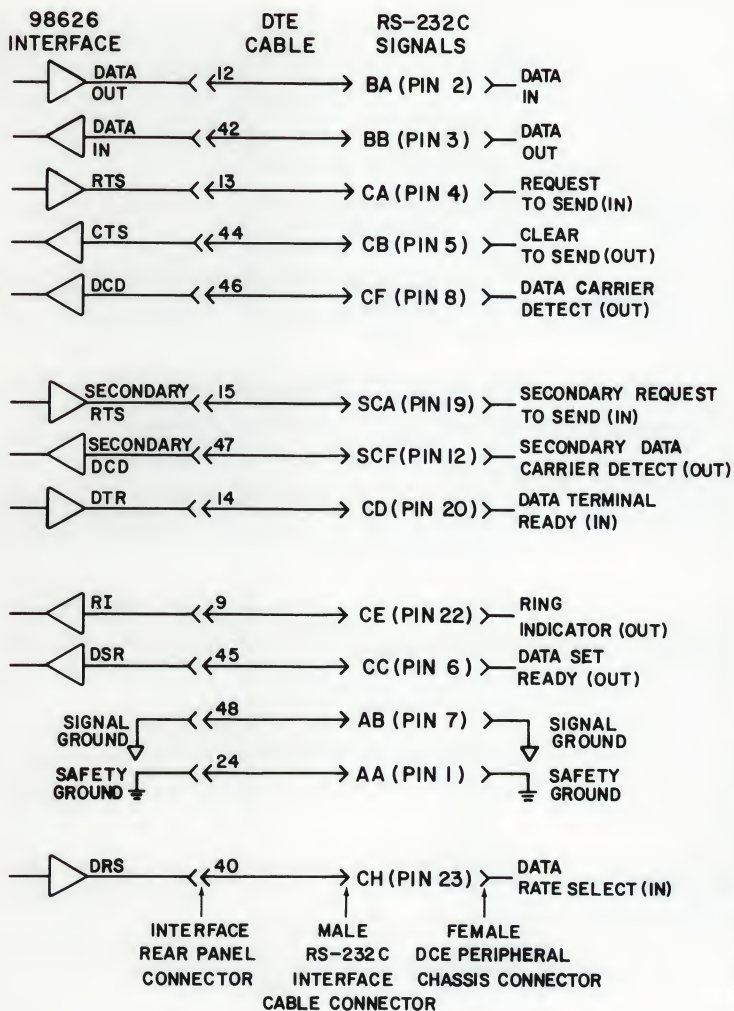
---

#### Note

Not all RS-232C devices are wired the same. To ensure proper operation, you must know whether the peripheral device is wired as DTE or DCE. The interface cable option and associated adapter cable, if needed, must be configured to properly mate with the female DTE chassis connector.

---

The following schematic diagram shows the input and output signals for the serial interface and how they are connected to a DCE peripheral.



DCE Interface  
Signals to and  
from Peripheral

NOTE: Some DCE  
peripherals may not  
provide for all the  
signal lines shown.

Figure 13-3. DTE Cable Interconnection Diagram

This diagram shows an HP 13242M adapter cable connected to a DCE interface cable and a DTE peripheral. Note that RTS is connected to CTS in the DCE cable. If your peripheral uses RTS/CTS handshaking, a different adapter cable must be used with the appropriate DTE or DCE interface cable option.

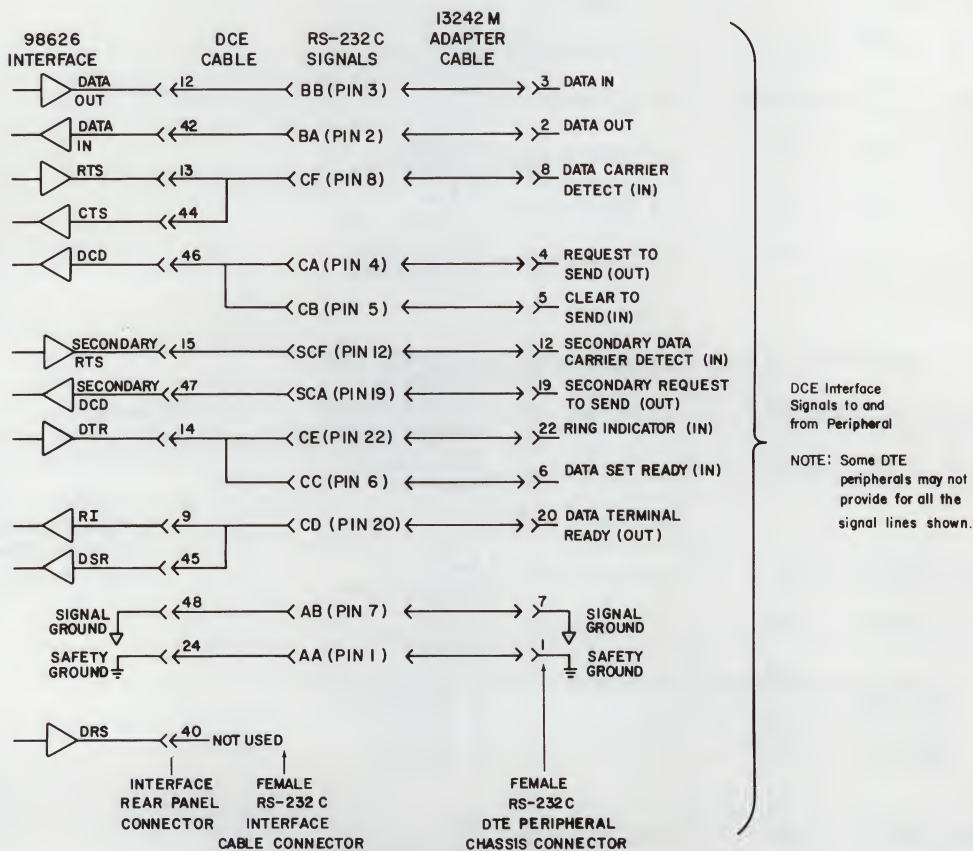


Figure 13-4. DCE Cable Interconnection Diagram



## RS-232C / CCITT V24

The following table provides information about each data communications interface function. The pin assignments are also shown. Not all functions provided by RS-232C standard are implemented. The functions denoted with a \* *are* implemented.

Table 13-7. RS-232C/CCITT V.24<sup>1</sup>

RS-232C	CCITT V24	Signal Name
*Pin 1	101	PROTECTIVE GROUND. Electrical equipment frame and ac power ground.
*Pin 2	103	TRANSMITTED DATA. Data originated by the terminal to be transmitted via the sending modem.
*Pin 3	104	RECEIVED DATA. Data from the receiving modem in response to analog signals transmitted from the sending modem.
*Pin 4	105	REQUEST TO SEND. Indicates to the sending modem that the terminal is ready to transmit data.
*Pin 5	106	CLEAR TO SEND. Indicates to the terminal that its modem is ready to transmit data.
*Pin 6	107	DATA SET READY. Indicates to the terminal that its modem is not in a test mode and that modem power is ON.
*Pin 7	102	SIGNAL GROUND. Establishes common reference between the modem and the terminal.
*Pin 8	109	DATA CARRIER DETECT. Indicates to the terminal that its modem is receiving carrier signals from the sending modem.
Pin 9		Reserved for test.
Pin 10		Reserved for test.
Pin 11		Unassigned.
*Pin 12	122	SECONDARY DATA CARRIER DETECT. Indicates to the terminal that its modem is receiving secondary carrier signals from the sending modem.
Pin 13	121	SECONDARY CLEAR TO SEND. Indicates to the terminal that its modem is ready to transmit signals via the secondary channel.

Note that the signals on pins 2, 3, and 7 above are commonly used for 3 wire (no modem) links.

<sup>1</sup> International Telephone and Telegraph Consultative Committee European standard.

**Table 13-7. RS-232C/CCITT V24 (continued)**

RS-232C	CCITT V24	Signal Name
Pin 14	118	SECONDARY TRANSMITTED DATA. Data from the terminal to be transmitted by the sending modem's channel.
*Pin 15	114	TRANSMITTER SIGNAL ELEMENT TIMING. Signal from the modem to the transmitting terminal to provide signal element timing information.
Pin 16	119	SECONDARY RECEIVED DATA. Data from the modem's secondary channel in response to analog signals transmitted from the sending modem.
*Pin 17	115	RECEIVER SIGNAL ELEMENT TIMING. Signal to the receiving terminal to provide signal element timing information.
Pin 18		Unassigned.
*Pin 19	120	SECONDARY REQUEST TO SEND. Indicates to the modem that the sending terminal is ready to transmit data via the secondary channel.
*Pin 20	108.2	DATA TERMINAL READY. Indicates to the modem that the associated terminal is ready to receive and transmit data.
Pin 21	110	SIGNAL QUALITY DETECTOR. Signal from the modem telling whether a defined error rate in the received data has been exceeded.
*Pin 22	125	RING INDICATOR. Signal from the modem indicating that a ringing signal is being received over the line.
*Pin 23	111	DATA SIGNAL RATE SELECTOR. Selects one of two signaling rates in modems having two rates.
*Pin 24	113	TRANSMIT SIGNAL ELEMENT TIMING. Transmit clock provided by the terminal.
Pin 25		Unassigned.

---

## HP 98626 and HP 98644 Serial Interface STATUS and CONTROL Registers

**General Notes:** Most Control registers accept values in the range of zero through 255. Some registers accept only specified values as indicated, or higher values for baud rate settings. Values less than zero are not accepted. Higher-order bits not needed by the interface are discarded if the specified value exceeds the valid range.

Reset value is the default value used by the interface after a reset or power-up until the value is overridden by a CONTROL statement.

The STATUS and CONTROL register information contained in this section applies only to BASIC/UX. BASIC/WS and BASIC/DOS support additional STATUS and CONTROL registers not covered in this section.

### **STATUS Register 0**      Card Identification

Value returned: 2 indicates a 98626 (if 130 is returned, the Remote jumper wire has been removed from the interface card); 66 indicates a 98644 (194 if the Remote jumper has been removed).

### **CONTROL Register 0**      Interface Reset

Any value from 1 thru 255 resets the card. Execution is immediate; any data transfers in process are aborted and any buffered data is destroyed. A value of 0 causes no action.

### **STATUS Register 1**      Interrupt Status (not supported)

### **CONTROL Register 1**      Send BREAK

Any non-zero value causes a BREAK to be sent.



**STATUS Register 2****Interface Activity Status**

Bit 7 thru 4 are not used.

Bit 3 set: Error condition. Handshake ended with an escape.

Bit 2 set: Handshake in progress. This occurs only during multi-line function calls.

Bit 1 set: Firmware interrupts enabled (ENABLE INTR active for this select code). This bit is **not** supported on BASIC/UX.

Bit 0 set: TRANSFER in progress.

**STATUS Register 3****Current Baud Rate**

Returns one of the values listed under CONTROL Register 3.

**CONTROL Register 3****Set New Baud Rate**

Use any one of the following values:

50	150	1200	4800
75	200	1800	7200
110	300	2400	9600
134	600	3600	19200

From 25 to 28800, the value will be rounded. Any value outside this range gives an error.

**STATUS Register 4**      Current Character Format

See CONTROL Register 4 for function of individual bits.

**CONTROL Register 4**    Set New Character Format

**Table 13-8. Character Format and Parity Settings**

Handshake (Bits 7&6)	Parity Sense <sup>1</sup> (Bits <sup>2</sup> 5&4)	Par. Enable (Bit <sup>2</sup> 3)	Stop Bits (Bit <sup>2</sup> 2)	Char. Length (Bits <sup>2</sup> 1&0)
00 no-op	00 ODD parity	0 Disabled	0 1 stop bit	00 5 bits/char
01 Xon/Xoff	01 EVEN parity	1 Enabled	1 2 stop bits	01 6 bits/char
Bidirectional	10 Unsupported			10 7 bits/char
10 Unsupported	11 Unsupported			11 8 bits/char
11 Handshake Disabled				

**STATUS Register 5**      Current Status of Modem Control Lines

Returns CURRENT line state values. See CONTROL Register 5 for function of each bit.

00000011

<sup>1</sup> Parity sense valid only if parity is enabled (bit 3=1). If parity is disabled, parity sense is meaningless.

<sup>2</sup> These bits correspond to equivalent switch settings on the HP 98626 and HP 98644 serial interface cards. A 1 is the same as set.

**CONTROL Register 5**    Set Modem Control Line States

Sets Modem Control lines or interface state as follows:

Bit 2 set:    Set Data Rate Select modem line to active state.

Bit 1 set:    Force Request-to-Send modem line to fixed active state.

Bit 1 clear: Toggle RTS line as in normal OUTPUT operations.

Bit 0 set:    Force Data Terminal Ready modem line to fixed active state.

Bit 0 clear: Toggle DTR line as in normal OUTPUT and ENTER operations.

**STATUS Register 6**    Data In (not supported)

**CONTROL Register 6**    Data Out (not supported)

**STATUS Register 7**    Optional Receiver/Driver Status (not supported)

**CONTROL Register 7**    Set New Optional Driver States (not supported)

**STATUS Register 8**    Current Interrupt Enable Mask (not supported)

**STATUS Register 9**    Cause of Current Interrupt (not supported)



**STATUS Register 10**      UART Status (not supported)

**STATUS Register 11**      Modem Status

Bit set indicates that the specified modem line or condition is active, and bit clear indicates that the specified modem line is not active. The default settings are given with each bit.

Bit 7: Data Carrier Detect (DCD) modem line active.

Bit 6: Ring Indicator (RI) modem line active.

Bit 5: Data Set Ready (DSR) modem line active.

Bit 4: Clear-to-Send (CTS) modem line active.

Bit 3: Change in DCD line state detected  
(not supported).

Bit 2: RI modem line changed from true to false

Bit 1: Change in DSR line state detected  
(not supported).

Bit 0: Change in CTS line state detected  
(not supported).

**STATUS Register 12**      Modem handshake status (not supported)

**CONTROL Register 12**      Modem handshake control (not supported)

**STATUS Register 13**      Read "SCRATCH A default" baud rate

Returns the baud rate that will be restored whenever SCRATCH A is executed (same bit-definitions as STATUS register 3).

**CONTROL Register 13** Set "SCRATCH A default" baud rate

Sets both the "current" and the "default" baud rate that will be restored whenever SCRATCH A is executed (same bit-definitions as CONTROL register 3).

**STATUS Register 14** Read "SCRATCH A default" character format

Returns the character format parameters that will be restored whenever SCRATCH A is executed (same bit-definitions as STATUS register 4).

**CONTROL Register 14** Set "SCRATCH A default" character format

Sets both the "current" and the "default" character format parameters that will be restored whenever SCRATCH A is executed (same bit-definitions as CONTROL register 4).

---

## HP 98644 Interface Differences

The HP 98644 RS-232 Serial Interface is nearly identical to the HP 98626 RS-232 Serial Interface. This section describes the few differences between them.

### Hardware Differences

The differences in the hardware of the two cards occur in the following areas:

- Card ID register contains 66 (rather than 2) or 194 if the Remote jumper on the HP 98644 interface card has been removed.
- There are no optional driver and receiver lines.
- There are fewer configuration switches (there are no Baud Rate or Line Control switches).
- There is a 25-pin coverplate connector (instead of 50).
- There are different cables available.

### Card ID Register

The default card ID for the HP 98644 interface is 66, and the default card ID for the HP 98626 is 2.

---

#### Note

HP 98644 cards are logged as HP 98626 interfaces while booting machines with Boot ROM 3.0 (and earlier versions). This is not a problem, because the BASIC recognizes the 98644 card properly.

You can also change the card ID to 2 (to make it look like a 98626) by cutting a jumper on the card. See the 98644's installation manual for details.

---

See the following "BASIC Differences" section for details of how to read this register with software.



### **Optional Driver Receiver Circuits**

On the 98626 interface, there are two optional driver lines (OCD3 and OCD4) and two optional receiver lines (OCR2 and OCR3). These lines are **not** implemented on the 98644 interface.

### **Configuration Switches**

The 98644 card does not implement the following configuration switches on the card:

- Baud Rate
- Line Control (character length, parity, etc.)

These operating parameters are set in the same manner as the HP 98626 interface card. See the previous section "Serial Configuration for HP-UX" for details.

### Coverplate Connector

The connector on the 98644 interface's coverplate is set up for DTE (Data Terminal Equipment) applications; it has a 25-pin, female, D-series connector (the connector on the 98626 is a 50-pin connector). Here are the pin designators for the connector.

**Table 13-9. Coverplate Connector Pin Designators**

Pin	Signal Description
1	Safety Ground
2	Transmitted Data
3	Received Data
4	Request to Send
5	Clear to Send
6	Data Set Ready
7	Signal Ground
8	Carrier Detect
9	not used
10	not used
11	not used
12	not used
13	not used
14	not used
15	not used
16	not used
17	not used
18	not used
19	not used
20	Data Terminal Ready
21	not used
22	Ring Indicator
23	Data Rate Select
24	not used
25	not used

## Cables

You can use standard RS-232C compatible cables, as long as the signal lines are connected properly. Here are cables available from HP Computer Supplies Operation.

**Table 13-10. Available RS-232C-Compatible Cables**

HP Product Number	Description
13242N	Modem cable (male to male)
13242G	DTE cable (male to male, with pins 2 and 3 reversed)
13242H	DCE cable (male to <i>female</i> , with pins 2 and 3 reversed)

## BASIC Differences

The only differences between programming these two interfaces with the BASIC system are in the register definitions given in this section. See the "Summary of RS-232 Serial STATUS and CONTROL Registers" section for further details.

### Card ID Register

The card ID register is Status register 0. It will contain a value of 66 if the interface is a 98644. (It will contain 2 if the card ID jumper has been cut.) If the REMOTE jumper has been removed, then the value returned will be 194 (=128+66) or 130 (=128+2).

---

## Series 300

### Built-In 98644 Interface Differences

The differences between the separate HP 98644 RS-232C serial interface and the built-in 98644-like interface of Series 300 computers are as follows:

- There are no "Select Code" switches (the select code is hard-wired to 9).
- There are no "Interrupt Level" switches (the interrupt level is hard-wired to 5).

There are no differences in programming these interfaces with the BASIC system.



# Table of Contents

---

## Chapter 14: Datacomm Interfaces

Prerequisites .....	14-2
Asynchronous Communication Protocol .....	14-3
Data Transfers Between Computer and Interface .....	14-4
Outbound Control Blocks .....	14-4
Inbound Control Blocks .....	14-5
Outbound Data Messages .....	14-5
Inbound Data Messages .....	14-6
Overview of Datacomm Programming .....	14-7
Establishing the Connection .....	14-8
Determining Protocol and Link Operating Parameters .....	14-8
Datacomm Configuration for BASIC/UX .....	14-9
Resetting the Datacomm Interface .....	14-11
Datacomm Options for Async Communications .....	14-12
Connecting to the Line .....	14-15
Connection Procedure for Hayes-Compatible Modems .....	14-16
Datacomm Error Recovery .....	14-18
Datacomm Error Detection and Program Recovery .....	14-19
Cable and Adapter Options and Functions .....	14-20
DTE and DCE Cable Options .....	14-20
Optional Circuit Driver/Receiver Functions .....	14-21
RS-232C/CCITT V24 .....	14-23
The HP 98642 4-Channel Multiplexer .....	14-25
Specifics on the HP 98642 4-Channel Multiplexer .....	14-25
Using the HP 98642 4-Channel Multiplexer .....	14-25
Keywords Used by the HP 98642 4-Channel Multiplexer .....	14-26
HP 98628 and HP 98642 Datacomm Interface Status and Control Registers .....	14-28



## Datcomm Interfaces

The HP 98628 and HP 98642 Data Communications Interfaces enable your desktop computer to communicate with any device that is compatible with standard asynchronous data communication protocols. Devices can include modems or equipment with standard RS-232C links. Because the HP 98628 and HP 98642 Data Communications Interface cards have only a few differences, this chapter will deal mainly with the HP 98628 interface card. Information on differences between these two data communications cards can be found in the section "The HP 98642 4-Channel Multiplexer."

This chapter only discusses asynchronous protocol. Data link protocol is not supported by BASIC/UX.

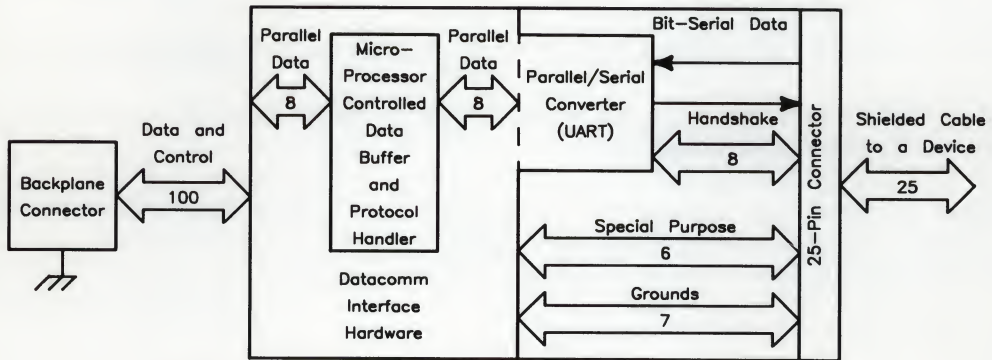


Figure 14-1. Block Diagram of the Datcomm Interface



## Prerequisites

It is assumed that you are familiar with the information presented in Data Communication Basics, and that you understand data communication hardware well enough to determine your needs when configuring the datacomm link. Configuration parameters include such items as half/full duplex, handshake, and timeout requirements. If you have any questions concerning equipment installation or interconnection, consult the appropriate interface or adapter installation manuals.

The datacomm interface supports several cable and adapter options. They include:

- RS-232C Interface cable and connector wired for operation with data communication equipment (male cable connector) or with data terminal equipment (female cable connector).
- HP 13265A Modem for asynchronous connections up to 300 baud, including built-in autodial capability<sup>1</sup>.
- HP 13266A Current Loop Adapter for use with current loop links or devices.

Some of the information contained in this chapter pertains directly to certain of these devices in specific applications.

Before you begin datacomm operation, be sure all interfaces, cables, connectors, and equipment have been properly plugged in. Power must be on for all devices that are to be used. Consult applicable installation manuals if necessary.

---

<sup>1</sup> The HP 13265A modem is compatible with Bell 103 and Bell 113 Modems, and is approved for use in the USA and Canada. Most other countries do not allow use of user-owned modems. Contact your local HP Sales and Service office for information about local regulations.

## Asynchronous Communication Protocol

Asynchronous protocol is the only protocol supported by BASIC/UX. Asynchronous data communication is the most widely used protocol, especially in applications where high data integrity is not mandatory. Data is transmitted, one character at a time, with each character being treated as an individual message. Start and stop bits are used to maintain timing coordination between the receiver and transmitter. A parity bit is sometimes included to detect character transmission errors. Asynchronous character format is as follows: Each character consists of a start bit, 5 to 8 data bits, an optional parity bit, and 1, 1.5, or 2 stop bits, with an optional time gap before the beginning of the next character. The total time from the beginning of one start bit to the beginning of the next is called a character frame.

Parity options include:

- NONE No parity bit is included.
- ODD Parity set if EVEN number of "1"s in character bits.
- EVEN Parity set if ODD number of "1"s in character bits.
- ONE Not supported.
- ZERO Not supported.

Here is a simple diagram showing the structure of an asynchronous character and its relationship to previous and succeeding characters:

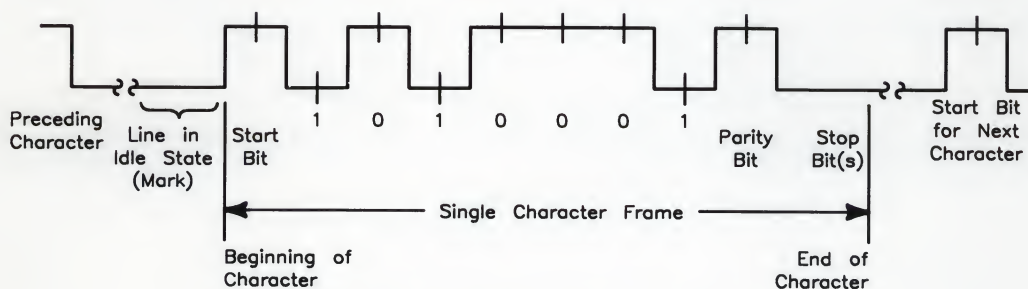


Figure 14-2. Structure of Asynchronous Character

---

## Data Transfers Between Computer and Interface

Data transfers between your desktop computer and its datacomm interface involve two message types: control blocks and data. Control blocks contain information sent to and received from the interface regarding its operation. Data is sent to and received from a remote device through the interface. Control blocks are not sent to or received from remote devices. Both types are encountered in both output and input operations as follows:

- Outbound control blocks are created by CONTROL statements.
- Outbound data messages are created by OUTPUT statements.
- Inbound control blocks are created by certain protocol operations such as Data Link block boundaries, or Async prompt, end-of-line, parity/framing error, or break detection.
- Inbound data messages are created by the interface as messages are received from the remote. They are transferred to BASIC by ENTER statements.

### Outbound Control Blocks

Outbound control blocks are messages from your computer to the datacomm interface that contain interface control information. They are usually generated by CONTROL statements, although OUTPUT...END creates a control block that terminates a given Async transmission or forces a block to be sent on the Data Link. Outbound control blocks are serially queued with data, and executed by the interface in the same order as created by BASIC. The single exception to the queued control block rule is when a non-zero value is output to Control Register 0 (Interface Reset) which is executed immediately.



---

### Note

When an interface card reset is executed by use of a CONTROL statement, the control block that results is transmitted directly to the interface. It is not queued up, so any previously queued data and control blocks are destroyed. To prevent loss of data, be sure that all queued messages have been sent before resetting the datacomm interface. Status Register 38 returns a value of 1 when the outbound queue is empty. Otherwise, its value is 0. To prevent loss of inbound data, Status Register 5 must return a value of zero prior to reset.

---

### Inbound Control Blocks

Inbound control blocks are messages from the interface to the computer that identify protocol control information. Refer to the *BASIC Language Reference* Control and Status Register section for details about register contents for various control block types.

For Async applications, terminal emulator programs usually use prompt and end-of-line control blocks. Use of other functions such as break or error detection depend on the requirements of the individual application.

### Outbound Data Messages

Outbound data messages are created when an OUTPUT statement is executed. Here is a short summary of how OUTPUT parameters can affect datacomm operation. Data is transmitted directly from the outbound queue. When operating in half-duplex, OUTPUT...END causes the interface to turn the line around and allow the remote device to send information back (line turn-around is initiated when the interface sets the Request-to-send line low). OUTPUT...END has no effect when operating in full duplex.

Unless a semicolon or END appears at the end of a free-field OUTPUT statement, an EOL sequence is automatically sent at the end of the data. The EOL sequence is also suppressed by using the appropriate IMAGE specifier in an OUTPUT statement. For further information, see the chapter called "Outputting Data."



## **Inbound Data Messages**

Inbound data messages are created by the datacomm interface as information is received from the remote. ENTER statements are terminated when a control block is encountered or the input variable is filled.

With this brief introduction to the data communications capabilities of the HP 98628 Datacomm Interface, you are ready to begin programming your desktop computer for datacomm operation. The next section of this chapter introduces BASIC datacomm programming techniques using simple terminal emulator examples that can be readily expanded into much more sophisticated datacomm programs.

---

## Overview of Datacomm Programming

Your desktop computer uses four BASIC statements for data communication with remote computers, terminals, and other peripheral devices. Datacomm programs include part or all of the following elements:

- CONTROL statements to configure the datacomm link and establish the connection.
- OUTPUT and ENTER statements to transfer information.
- STATUS statements to monitor operation.
- CONTROL statements to alter link parameters during the session, if needed for unusual applications.
- OUTPUT and ENTER statements to transfer additional information.
- A CONTROL statement to disconnect at the end of the session.

---

# Establishing the Connection

## Determining Protocol and Link Operating Parameters

Before information can be successfully transferred between two devices, a communication link must be established. You must include the necessary protocol parameters to ensure compatibility between the communicating machines. To determine the proper parameters for your application, answer the following questions:

- Is a modem connection being used? What handshake provisions are required?
- Is half-duplex or full-duplex line protocol being used?
- What line speed (baud rate) is being used?
- How many bits (excluding start, stop, and parity bits) are included in each character?
- What parity is being used: none, odd, even, always zero, or always one?
- How many stop bits are required on each character you transmit?
- What line terminator should you use on each outgoing line?
- How much time gap is required between characters (usually 0)?
- What prompt, if any, is received from the remote device when it is ready for more data?
- What line terminator, if any, is sent at the end of each incoming line?

All these parameters are configured under program control by use of `CONTROL` statements. Alternately, default values for line speed, modem handshake, and parity can be set as described in the section "Datacomm Configuration for BASIC/UX." Other default parameters are preset by the datacomm interface to accommodate common configurations. You can use the defaults, or you can override them with `CONTROL` statements for program clarity and immunity to HP-UX defaults. Default Control Register values are shown in the "Interface Register" section in the back of the *BASIC Language Reference* for your desktop computer.

## Datacomm Configuration for BASIC/UX

There is no capability in BASIC/UX for reading the hardware switches on either the HP 98628 Datacomm Interface card or the HP 98642 4-Channel Multiplexer card. Therefore, BASIC/UX provides two methods for configuring modem control options:

- The **stty** command from the HP-UX environment.
- The keyword **CONTROL** and registers directly related to the modem control options.

Of the two methods mentioned above the best one to use is the **stty** command while in the HP-UX environment. The reason for this is any modem control options set by using the keyword **CONTROL** are lost when you leave BASIC/UX. However, if you prefer to change these options while in the BASIC/UX environment, then read the subsequent section "Datacomm Options for Async Communications."

This section deals with the first method mentioned above which is the use of the **stty** command from the HP-UX environment.

### Defaults for the Serial Interface

When HP-UX is being booted, the defaults for all Datacomm Interfaces are:

Baud rate	300
Bits per character	7
Parity	Odd
Stop bits	2

The above values are used by BASIC/UX as defaults, unless configured as explained in the next section.

Some common datacomm interface configuration settings are:

Baud rate to	9600
Bits per character to	8
Parity to	Odd and disabled
Stop bits to	1



## Configuring a Datacomm Interface for BASIC/UX

To configure your datacomm interface with the values mentioned in the previous section, you can execute the following HP-UX command before entering BASIC/UX:

```
/bin/stty 9600 cs8 -parenb parodd -cstopb < /dev/rmb/dcommnn
```

where:

**9600** is the baud rate. The following are baud rates you can use with the *stty* command:

50	150	1200	4800
75	200	1800	7200
110	300	2400	9600
134	600	3600	19200

**cs8** is the number of bits per character. In the case of this example, the number of bits per character is 8. Other character lengths can be set using **cs5**, **cs6**, or **cs7** for 5, 6, or 7 bits per character respectively.

**-parenb** disables parity generation and detection. Removing the minus sign that is prefixed to this *stty* option causes parity generation and detection to be enabled.

**parodd** selects odd parity. Prefixing the minus sign to this *stty* option selects even parity.

**-cstopb** causes one stop bit per character to be used. Removing the minus sign that is prefixed to this *stty* option causes two stop bits per character to be used.

**< /dev/rmb/dcommnn** assigns the *stty* options to the serial interface located at select code number *nn*.

For more information on *stty* options, see the *HP-UX Language Reference*.

## Resetting the Datacomm Interface

Before you establish a connection, the datacomm interface must be in a known state. **The datacomm interface does not automatically disconnect from the datacomm link when the computer reaches the end of a program.** To prevent potential problems caused by unknown link conditions left over from a previous session, it is a good practice to reset the interface card at the beginning of your program before you start configuring the datacomm connection. Resetting the card causes it to disconnect from the line and return to a known set of initial conditions (see the previous section "Datacomm Configuration for HP-UX").

In the following example, a numeric variable is used to define the select code. The second statement resets the card after the select code has been defined.

```
1110    Sc=20           ! Set select code to 20.
1160    CONTROL Sc,0;1 ! Reset the card to disconnect from line.
```

You are now ready to program datacomm options that are related to the Async protocol. In applications where defaults are used, the options are very simple. The following example shows how to set up datacomm options for Async protocol.

## Datacomm Options for Async Communications

This section explains how to configure the datacomm interface for asynchronous data communication. The example used shows how to set up all configurable options without considering default values. Some statements in the example are redundant because they override interface defaults having the same value. Others may or may not be redundant because they override HP-UX default configurations. The remaining statements are necessary because they override the default values, replacing them with non-default values required for proper operation of the example program. If you are not familiar with Asynchronous protocol, consult the section on protocol for the needed background information.

The following program lines set up all the CONTROL register options (a 300-baud connection to an HP 1000 is assumed):

```
→ 1310 CONTROL Sc,20;7      ! Transmit speed = 300 baud.
    1330 CONTROL Sc,22;5      ! DC1/DC3 (as terminal & host) handshake.
→ 1340 CONTROL Sc,34;2      ! Seven bits per character.
→ 1350 CONTROL Sc,35;0      ! One stop bit.
→ 1360 CONTROL Sc,36;1      ! Odd parity.
```

→: May be redundant. Overrides configuration switch option.

Refer to the Control Register tables in the back of the *BASIC Language Reference* as you examine the CONTROL statements. The paragraphs which follow explain register functions and how to configure them.

### Datacomm Line Timeouts

Registers 16-19 set timeout values to force an automatic disconnect from the datacomm link when certain time limits are exceeded. For most applications, the default values are adequate. A value of zero disables the timeout for any register where it is used. Each register accepts values of 0 thru 255; units vary with the register function.

- Register 16 (Connection timeout) sets the time limit (in seconds) allowed for connecting to the remote device. It is useful for aborting unsuccessful attempts to dial up a remote computer using public telephone networks.
- Register 17 (No Activity timeout) sets an automatic disconnect caused by no datacomm activity for the specified number of minutes. Default value is determined by default handshake switch setting. Default is not affected by CONTROL statements to Control Register 23 (hardware handshake).



- Register 18 (Lost Carrier timeout) disconnects when:

**Full Duplex:** Data Set Ready (Data Mode) or Data Carrier Detect go false, or

**Half Duplex:** Data Set Ready goes false,

indicating that the carrier from the remote modem has disappeared from the line. Value is in multiples of 10 milliseconds.

- Register 19 (Transmit timeout) disconnects when a loss-of-clock occurs or a clear-to-send (CTS) is not returned by the modem within the specified number of seconds.

### Line Speed (Baud Rate)

The transmit and receive line speed(s) are set by Control Registers 20 and 21, respectively. Each is independent of the other, and they are not required to have identical values. The following baud rates are available for Async communication:

**Table 14-1. Async Baud Rates**

Register Value	Baud Rate	Register Value	Baud Rate	Register Value	Baud Rate	Register Value	Baud Rate
0	0 <sup>1</sup>	4	134	8	600 <sup>2</sup>	12	3600
1	50	5	150 <sup>2</sup>	9	1200 <sup>2</sup>	13	4800 <sup>2</sup>
2	75	6	200	10	1800	14	9600 <sup>2</sup>
3	110 <sup>2</sup>	7	300 <sup>2</sup>	11	2400 <sup>2</sup>	15	19 200

All configurable line speeds are available to CONTROL Registers 20 and 21. Defaults are set as described in the previous section "Datacomm Configuration for BASIC/UX."

<sup>1</sup> An external clock must be provided for this option.

<sup>2</sup> These speeds can be programmed using the default switches on the interface card. Other speeds are accessed by CONTROL statements. (The HP 13265A Modem can be operated up to 300 baud.)



## Handshake

Register 22 configures software handshake parameters and specifies which of the participants is allowed to transmit while the other agrees to receive until the exchange is reversed. **Options** include:

- **No handshake**, commonly used with connections to non-interactive devices such as printers.
- **DC1/DC3 handshake** with the desktop computer configured either as a host or a terminal.
- **DC1/DC3 handshake** with the desktop computer as both a host AND a terminal. This option simplifies communication between two desktop computers.

## Character Format Definition

Registers 34 through 36 are used to define the character format for transmitted and incoming data.

- Register 34 sets the character length to 5, 6, 7, or 8 bits. The value used is the number of bits per character minus five (0=5 bits, 3=8 bits). When 8-bit format is specified, parity must be Odd, Even, or None (parity "1" or "0" cannot be used).
- Register 35 specifies the number of stop bits sent with each character. Values of 0 or 2 are used to select 1 or 2 stop bits, respectively.
- Register 36 specifies the parity to be used. Options include:

**Table 14-2. Parity Options**

Register Value	Parity	Result
0	None	Characters are sent with no parity bit. No parity checks are made on incoming data.
1	Odd <sup>1</sup>	Parity bit is set if there is an EVEN number of ones in the character code. Incoming characters are also checked for odd parity.
2	Even <sup>1</sup>	Parity bit is set if there is an ODD number of ones in the character code.
3	0	Unsupported on BASIC/UX.
4	1	Unsupported on BASIC/UX.

<sup>1</sup> Parity sense is based on the number of ones in the character including the parity bit. An EVEN number of ones in the character, plus the parity bit set produces an ODD parity. An ODD number of ones in the character plus the parity bit set produces an EVEN parity.

## **Connecting to the Line**

Interface configuration is now complete. You are ready to begin connecting to the datacomm line. The exact procedure used to connect to the line varies slightly, depending on the type of link being used. Before you connect, you must know what the link requirements are, including dialing procedures, if any.

### **Switched (Public) Telephone Links**

When you are using a public or switched telecommunications link, the modem connection between computers must be established. The HP 13265A Modem can be used in any Async application that requires a Bell 103- or Bell 113-compatible modem operating at up to 300 baud line speed. However, the HP 13265A Modem is not suitable for data rates exceeding 300 baud. For higher baud rates, use a modem that is compatible with the one at the remote computer site.

### **Private Telecommunications Links**

Private (leased) links require modems unless the link is short enough for direct connection (up to 50 feet, depending on line speed). The HP 13265A Modem can be used at data rates up to 300 baud. For higher speeds, a different modem must be used.

### **Direct Connection Links**

For short distances, a direct connection may be used without modems or adapters, provided both machines use compatible interfaces. Async connections normally use RS-232C interfaces.

## Connection Procedure for Hayes-Compatible Modems

Now that the CONTROL registers are set up for the Datacomm card (refer back to the section "Datacomm Options for Async Communications"), you are ready to initiate the connection. By OUTPUTing a dialing command to your modem, the connection will automatically be made. Here is an example of a simple dialing command:

```
1000  OUTPUT Sc;"ATDT 555-1234"
```

The OUTPUT statement contains several essential elements.

- AT is sent to get the attention of the modem and to tell it that you are going to be sending a command. AT must precede all commands sent to the modem.
- DT informs the modem that you want to dial using touch-tone dialing. Use "ATD" for rotary dialing.
- Numeric character sequences define the telephone number.

Once the connection is made, any data which you OUTPUT/ENTER to/from the Datacomm select code will be transmitted/received by the modem over the data line. To return to command mode, you need to OUTPUT an escape code. This escape code is usually "+++", but may vary with different modems. Once the escape code is sent, all data sent to the Datacomm select code will be treated by the modem as a command. Some of the commands are as follows:

AT	Attention
H	Hang-up
EO	Turn echo off
D	Dial
,	Pause
A/	Re-dial
O	Return on-line (Get back to data transmit/receive mode)

Note that all of the above commands must be preceded by "AT".

Refer to the user manual for your modem for a complete list of commands.

### Example Modem Session

A simple modem session may be as follows:

```
70   CONTROL Sc,0;1           ! Reset card.
80   CONTROL Sc,8;2           ! Set DTR line.
90   CONTROL Sc,20;11         ! Set rate to 2400.
100  CONTROL Sc,22;5          ! DC1/DC3 (as terminal & host) handshake.
110  CONTROL Sc,34;2          ! Seven bits per character.
120  CONTROL Sc,35;0          ! One stop bit.
130  CONTROL Sc,36;1          ! Odd parity.
140  OUTPUT Sc;"ATDT 555-1234" ! Establish the connection.
    .
    .
1990 ENTER Sc;Data$           ! Receive/Transmit.
2000 OUTPUT Sc;"Data"         ! Data over the connection.
    .
    .
3000 OUTPUT Sc;"++++";        ! Return to command mode (Send without
3010                               ! CR/LF).
3020 OUTPUT Sc;"ATH"          ! Hang-up!
3030 END
```



---

## Datacomm Error Recovery

When a forced disconnect terminates the connection, the interface is placed in a SUSPENDED state. The interface cannot be reconnected to the datacomm line when it is SUSPENDED. CLEAR, ABORT, and RESET are used to recover from the suspended state and resume normal card operation. Executing OUTPUT and CONTROL statements while the card is suspended places corresponding data and control block(s) in the transmit (outbound) queue and can continue to do so until the queue is filled, at which time the desktop computer operating system hangs. ENTER statements can be executed to retrieve data that was there prior to SUSPEND until the receive (inbound) queue is empty. Subsequent ENTER statements, if executed while the card is suspended, hang the computer.

To recover from a SUSPENDED interface, three programmable options are available, all of which destroy any existing data in the transmit and receive queues. They are:

- The CLEAR statement clears the receive and transmit queues. In addition, if the interface card is suspended, it disconnects the card from the datacomm line. If the card is not suspended, its connection state is not changed, but the queues are cleared.
- The ABORT statement is identical to the CLEAR statement, except that the interface card is unconditionally disconnected from the datacomm line.
- RESET interface (Control Register 0) clears all buffers and queues, and resets all CONTROL options to their power-up state.

A fourth (keyboard only) option is available. **SHIFT PAUSE** (or **RESET**) causes a hardware reset to be sent to ALL peripherals. This completely resets the datacomm interface to its power-up state.

---

## **Datacomm Error Detection and Program Recovery**

When a timeout or datacomm error occurs, an interrupt is generated by the interface card to BASIC. If an ON ERROR is active for that select code, the error is trapped and handled by the error routine specified by the ON ERROR statement. If no ON ERROR is active for that select code, the program is stopped at the end of the current line by the BASIC operating system, and an error message is sent to the PRINTER IS device.

When a datacomm error is trapped by an error routine, the routine must decide what to do about the problem. Since datacomm interface errors are not related to a specific program line, the ERRL function is always false, and ERRN returns the error number generated by the interface card. ERRL and ERRN are discussed in greater detail in the BASIC Programming Techniques manual for your desktop computer.

---

## Cable and Adapter Options and Functions

The HP 98628A Datacomm Interface is available with RS-232C DTE and DCE cable configurations, or it can be connected to various modems or adapters for other applications.

### DTE and DCE Cable Options

DTE and DCE cable options are designed to simplify connecting two desktop computers without the use of modems. The DTE cable (male RS-232 connector) is configured to make the datacomm interface look like standard data terminal equipment when it is connected to an RS-232C modem. The DCE cable (female RS-232 connector) is configured so that it eliminates the need for modems in a direct connection. When you connect two computers to each other in a direct non-modem connection, both datacomm interfaces are functionally identical. The DCE cable acts as an adapter so that both interfaces behave exactly as they would if they were connected to a pair of modems by means of DTE cables.

Several signal lines are rerouted in the DCE cable so that, in direct connections, outputs from one interface are connected to the corresponding inputs on the other interface. Certain outputs on each interface are also connected to inputs on the same card by "loop-back" connections in the DCE cable.

The schematic diagram in this section shows two datacomm interfaces directly connected through a DTE-DCE cable pair. Note that the DCE cable wiring complements the DTE cable so that output signals are properly routed to their respective destinations. Signal names at the RS-232C connector interface are the same as the signal names for the DTE interface. However, because the DCE cable adapts signal paths, the signal name at the RS-232C connector does not necessarily match the signal name at the DCE interface. Connector pin numbers are included in the diagram for your convenience.

**RS-232C DTE (male) Cable Signal Identification Tables**

Signal RS-232C	Signal V.24	Interface Pin#	RS-232C Pin#	Mnemonic	I/O	Function
AA	101	24	1	—	—	Safety Ground
BA	103	12	2	Out		Transmitted Data
BB	104	42	3	In		Received Data
CA	105	13	4	RTS	Out	Request to Send
CB	108	44	5	CTS	In	Clear to Send
CC	107	45	6	DSR	In	Data Set Ready
AB	102	48	7	—	—	Signal Ground
CF	109	46	8	DCD	In	Data Carrier Detect
SCF (OCR2)	122	47	12	SDCD	In	Secondary DCD
DB	114	41	15	In		DCE Transmit Timing
DD	115	43	17	In		DCE Receive Timing
SCA (OCD2)	120	15	19	SRTS	Out	Secondary RTS
CD	108.1	14	20	DTR	Out	Data Terminal Ready
CE (OCR1)	125	9	22	RI	In	Ring Indicator
CH (OCD1)	111	40	23	DRS	Out	Data Rate Select
DA	113	7	24	Out		Terminal Transmit Timing

### Optional Circuit Driver/Receiver Functions

Two optional drivers and receivers are used with the RS-232C cable options. Their functions are as follows:

#### Drivers

Name	Function
OCD1	Data Rate Select
OCD2	Secondary Request-to-send
OCD3	Not used
OCD4	Not used

#### Receivers

Name	Function
OCR1	Ring Indicator
OCR2	Secondary Data Carrier Detect



OCD2 is used for autodial pulsing in the HP 13265A Modem. None of the optional drivers and receivers are used for Data Link and Current Loop Adapters.

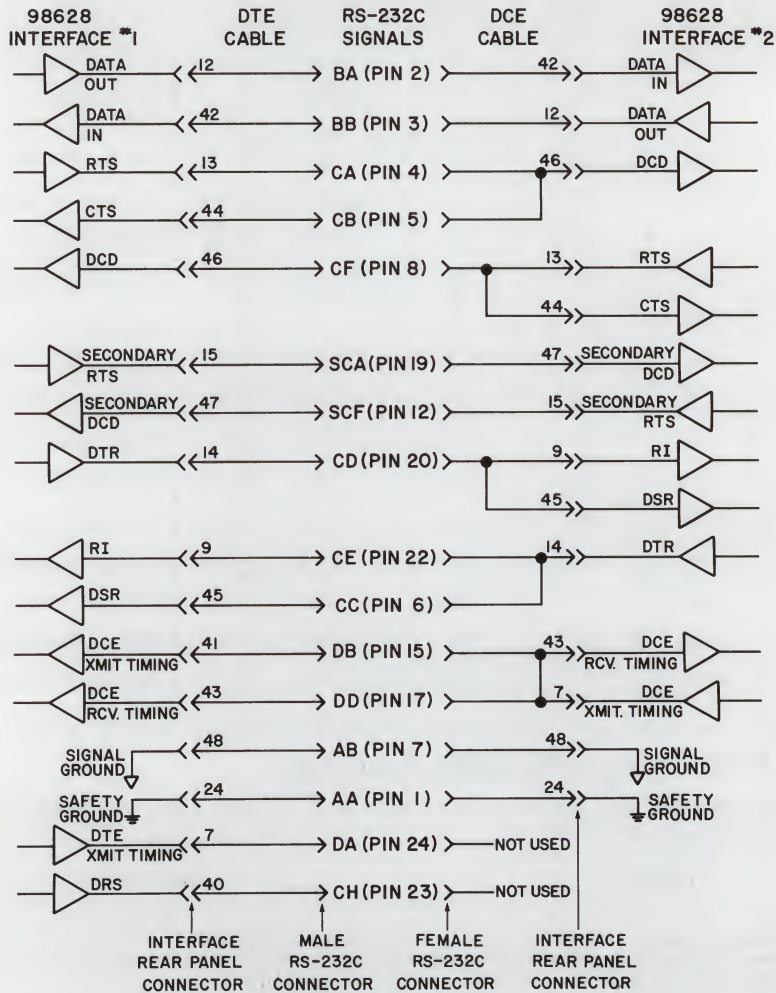


Figure 14-3. DTE/DCE Interface Cable Wiring

## RS-232C/CCITT V24

The following table provides information about each data communications interface function. The pin assignments are also shown. Not all of the functions provided by RS-232C are implemented. The functions denoted with an \* **are implemented**.

RS-232C/CCITT V24<sup>1</sup>

RS-232C	CCITT V24	Signal Name
*Pin 1	101	PROTECTIVE GROUND. Electrical equipment frame and ac power ground.
*Pin 2	103	TRANSMITTED DATA. Data originated by the terminal to be transmitted via the sending modem.
*Pin 3	104	RECEIVED DATA. Data from the receiving modem in response to analog signals transmitted from the sending modem.
*Pin 4	105	REQUEST TO SEND. Indicates to the sending modem that the terminal is ready to transmit data.
*Pin 5	106	CLEAR TO SEND. Indicates to the terminal that its modem is ready to transmit data.
*Pin 6	107	DATA SET READY. Indicates to the terminal that its modem is not in a test mode and that modem power is ON.
*Pin 7	102	SIGNAL GROUND. Establishes common reference between the modem and the terminal.
*Pin 8	109	DATA CARRIER DETECT. Indicates to the terminal that its modem is receiving carrier signals from the sending modem.
Pin 9		Reserved for test.
Pin 10		Reserved for test.
Pin 11		Unassigned.
*Pin 12	122	SECONDARY DATA CARRIER DETECT. Indicates to the terminal that its modem is receiving secondary carrier signals from the sending modem.
Pin 13	121	SECONDARY CLEAR TO SEND. Indicates to the terminal that its modem is ready to transmit signals via the secondary channel.

Note that the signals on pins 2, 3, and 7 above are commonly used for 3 wire (no modem) links.

<sup>1</sup> International Telephone and Telegraph Consultative Committee European standard.

### RS-232C/CCITT V24 (Cont'd)

RS-232C	CCITT V24	Signal Name
Pin 14	118	SECONDARY TRANSMITTED DATA. Data from the terminal to be transmitted by the sending modem's channel.
*Pin 15	114	TRANSMITTER SIGNAL ELEMENT TIMING. Signal from the modem to the transmitting terminal to provide signal element timing information.
Pin 16	119	SECONDARY RECEIVED DATA. Data from the modem's secondary channel in response to analog signals transmitted from the sending modem.
*Pin 17	115	RECEIVER SIGNAL ELEMENT TIMING. Signal to the receiving terminal to provide signal element timing information.
Pin 18		Unassigned.
*Pin 19	120	SECONDARY REQUEST TO SEND. Indicates to the modem that the sending terminal is ready to transmit data via the secondary channel.
*Pin 20	108.2	DATA TERMINAL READY. Indicates to the modem that the associated terminal is ready to receive and transmit data.
Pin 21	110	SIGNAL QUALITY DETECTOR. Signal from the modem telling whether a defined error rate in the received data has been exceeded.
*Pin 22	125	RING INDICATOR. Signal from the modem indicating that a ringing signal is being received over the line.
*Pin 23	111	DATA SIGNAL RATE SELECTOR. Selects one of two signaling rates in modems having two rates.
*Pin 24	113	TRANSMIT SIGNAL ELEMENT TIMING. Transmit clock provided by the terminal.
Pin 25		Unassigned.



---

## The HP 98642 4-Channel Multiplexer

This interface is supported by BASIC/UX for data communications. The topics covered in this section are:

- Specifics on the HP 98642 4-channel multiplexer
- Using the HP 98642 4-channel multiplexer
- Keywords used by the HP 98642 4-channel Multiplexer.

### Specifics on the HP 98642 4-Channel Multiplexer

The HP 98642 4-channel multiplexer has one port that functions the same as an HP 98628 Data Communication interface card and the remaining ports function the same as an HP 98628 interface card without the hardware handshaking or modem control. Each port has its own set of STATUS and CONTROL registers that are the same as those for an HP 98628 interface card. The datacomm interface multiplexer allows your computer to communicate with any device (such as a modem) that is compatible with standard asynchronous data communication protocols.

### Using the HP 98642 4-Channel Multiplexer

To communicate with another device using the multiplexer, you need to know the multiplexer's select code (for example, select code 16) and the primary address of each port on the multiplexer. The primary addresses associated with these ports are:

- |                |  |
|----------------|--|
| 00             | This port functions the same as an HP 98628 interface card.  |
| 01, 02, and 03 | These ports function the same as an HP 98628 interface card without hardware handshaking or modem control. |

The select code and primary address together form the device selector. If the HP 98642's select code is 16, then it will have four ports with device selectors: 1600, 1601, 1602, and 1603. The following example shows you how to use device selector 1600 to check for the current transmit timeout limit for a device connected to an HP 98642 multiplexer at port 1.

```
STATUS 1600,19;Tran_stat
```



## Keywords Used by the HP 98642 4-Channel Multiplexer

The following table contains a list of keyword examples used by the HP 98642 4-Channel Multiplexer and a description of the examples. Note that these same keywords can be used by the HP 98628 Data Communications interface.

**Table 14-3. Keywords Used by the HP 98642 Interface Card**

Keyword Examples	Example Description
ABORTIO @Source	Terminates any active transfer associated with the I/O path name. Assume that the I/O path name called @Source was assigned device selector 1600 (see the keyword ASSIGN given below).
ASSIGN 1600 TO @Source ASSIGN @Buffer TO BUFFER Real_buf(*)	Assigns the I/O path name called @Source to device selector 1600 and the I/O path name called @Buffer to a named buffer called Real_buf(*).
BREAK 1601	Directs the datacomm interface located at device selector 1601 to send a break sequence.
ENTER 1601 USING "K";Str_val\$	Reads character values from device selector 1601 into the string called Str_val\$.
CONTROL 1602,6;1	Causes a BREAK to be sent to device selector 1602.
OFF TIMEOUT 1603	Cancels event-initiated branches, from the interface at device selector 1603, previously defined and enabled by an ON TIMEOUT statement. An OFF TIMEOUT without any device selector disables all of the channels of the HP 98642 4-Channel Multiplexer.
ON TIMEOUT 1603,10 GOSUB Time_out	Defines and enables an event-initiated branch to be taken when an I/O timeout occurs on the specified interface located at device selector 1603.
OUTPUT 1602 USING "DD";22	Writes the integer value 22 to device selector 1602.

**Table 14-3. Keywords Used by the HP 98642 Interface Card (continued)**

Keyword Examples	Example Description
RESET 1603	Resets the interface located at device selector 1603.
STATUS 1602,0;Ret_val	Returns the card identification status of device selector 1602.
TRANSFER @Source TO @Buffer	Transfers the contents of the I/O name path called @Source to the I/O path name called @Buffer. Note that @Source was created using the keyword ASSIGN and device selector 1600, and @Buffer was created using the secondary keyword BUFFER with the keyword ASSIGN and the array called Real_buf(*) (see the keyword ASSIGN given above).

---

## HP 98628 and HP 98642 Datacomm Interface Status and Control Registers

**General Notes:** Control registers accept values in the range of zero through 255. Some registers require specified values, as indicated. Illegal values or values less than zero or greater than 255, cause ERROR 327.

Reset value, shown for various Control Registers, is the default value used by the interface after a reset or power-up until the value is overridden by a CONTROL statement.

- Status 0** Card Identification  
Value returned: 52 (if 180 is returned, check select code switch cluster and make sure switch R is ON).
- Control 0** Card Reset  
Any value, 1 thru 255, resets the card. Immediate execution. Data in queues is destroyed.
- Status 1** Hardware Interrupt Status (not supported)
- Status 2** Datacomm Activity  
Bit 0 set: TRANSFER in progress.  
Bit 1 set: Firmware interrupts enabled (ENABLE INTR active for this select code). This bit is not supported on BASIC/UX.  
Bit 2 set: Handshake in progress. Only during multi-line function calls.  
Bit 3 set: Handshake ended with an escape.
- Status 3** Current Protocol Identification: 1 = Async.
- Control 3** Protocol to be used after next card reset: CONTROL Sc,0;1 (not supported). 1 = Async Protocol
- Status 4** Cause of ON INTR program branch (not supported).
- Status 5** Inbound queue status (not supported)

**Control 5** Terminate Transmission (not supported)

**Status 6** Break status (not supported)

**Control 6** Send Break; causes a Break to be sent

Async Protocol: Transmit Break.

**Status 7** Modem receiver line states (values shown are for male cable connector option for connection to modems).

Bit 0: Data Mode (Data Set Ready) line

Bit 1: Receive ready (Data Carrier Detect line)

Bit 2: Clear-to-send (CTS) line

Bit 3: Incoming call (Ring Indicator line)

Bit 4: (not supported)

**Status 8** Returns modem driver line states.

**Control 8** Sets modem driver line states (values shown are for male cable connector option for connection to modems).

Bit 0: Request-to-send (RS or RTS) line 1 = line set (active)

Bit 1: Data Terminal Ready (DTR) line 0 = line clear (inactive)

Bit 2: Data Rate Select (DRS) line

Bits 3 through 7 are not supported on BASIC/UX

**Status 9** Returns control block TYPE if last ENTER terminated on a control block (not supported).

**Status 10** Returns control block MODE if last ENTER terminated on a control block (not supported).

**Status 11** Returns available outbound queue space (not supported)



- Status 12**    Datacomm Line connection status (not supported)
- Control 12**   Connects, initiates auto-dial sequence, and disconnects interface from datacomm line (not supported).
- Status 13**    Returns current ON INTR mask (not supported)
- Control 13**   Sets ON INTR mask (not supported)
- Status 14**    Returns current Control Block mask (not supported).
- Control 14**   Sets Control Block mask (not supported).
- Status 15**    Returns current modem line interrupt mask (not supported).
- Control 15**   Sets modem line interrupt mask (not supported).
- Status 16**    Returns current connection timeout limit (not supported).
- Control 16**   Sets Attempted Connection timeout limit (not supported).  
Acceptable values: 1 thru 255 seconds. 0=timeout disabled.  
**Reset Value=25 seconds**
- Status 17**    Returns current No Activity timeout limit (not supported).
- Control 17**   Sets No Activity timeout limit (not supported).  
Acceptable values: 1 thru 255 minutes. 0=timeout disabled.  
**Reset Value=10 minutes (disabled if Async, non-modem handshake).**

- Status 18** Returns current Lost Carrier timeout limit (not supported).
- Control 18** Sets Lost Carrier timeout limit in units of 10 ms (not supported).  
Acceptable values: 1 thru 255. 0=timeout disabled.  
**Reset Value=40** (400 milliseconds)
- Status 19** Returns current Transmit timeout limit (not supported).
- Control 19** Sets Transmit timeout limit (not supported). Note that loss of clock or CTS not returned by modem when transmission is attempted.  
Acceptable values: 1 thru 255.0=timeout disabled.  
**Reset Value=10 seconds**
- Status 20** Returns current transmission/receive speed (baud rate). See table for values.
- Control 20** Sets transmission speed (baud rate) as follows:

Register Value	Baud Rate	Register Value	Baud Rate
0	External Clock	8	600
*1	50	9	1200
*2	75	10	1800
*3	110	11	2400
*4	134	12	3600
*5	150	13	4800
*6	200	14	9600
7	300	15	19200

\* Note that this register is used to set the transmit and receive speed for asynchronous communication. The default value is defined by the HP-UX termio driver switches.

- Status 21** Same as Status register 20.
- Control 21** Same as Control register 20.

**Status 22** Protocol dependent. Returns protocol handshake type (Async) as specified by Control Register 22.

**Control 22** Defines the asynchronous protocol handshake type that is to be used.

Value	Handshake type
0	Protocol handshake disabled
1	ENQ/ACK with desktop computer as the host (ignored)
2	ENQ/ACK with desktop computer as a terminal (ignored)
3	DC1/DC3, desktop computer as host
4	DC1/DC3, desktop computer as a terminal
5	DC1/DC3, desktop computer as both host and terminal

**Status 23** Returns current hardware handshake type (not supported).

**Control 23** Sets hardware handshake (not supported).

**Status 24** Protocol dependent (not supported).

**Control 24** Protocol dependent (not supported).

**Status 25** Returns number of received errors since power up or reset (not supported).

**Status 26** Protocol dependent (not supported).

**Control 26** (not supported)  
(Async only)

**Status 27** Returns second protocol handshake character (not supported).  
(Async only)

**Control 27** (not supported)  
(Async only)

<b>Status 28</b> (Async only)	Returns number of characters in inbound End-of-line delimiter sequence (not supported).				
<b>Control 28</b> (Async only)	Sets number of characters in End-of-line delimiter sequence (not supported).				
<b>Status 29</b> (Async only)	Returns first End-of-line character (not supported).				
<b>Control 29</b> (Async only)	Sets first End-of-line character (not supported).				
<b>Status 30</b> (Async only)	Returns second End-of-line character (not supported).				
<b>Control 30</b> (Async only)	Sets second End-of-line character (not supported).				
<b>Status 31</b> (Async only)	Returns number of characters in Prompt sequence (not supported).				
<b>Control 31</b> (Async only)	Sets number of characters in Prompt sequence (not supported).				
<b>Status 32</b> (Async only)	Returns first character in Prompt sequence (not supported).				
<b>Control 32</b> (Async only)	Sets first character in Prompt sequence (not supported).				
<b>Status 33</b> (Async only)	Returns second character in Prompt sequence (not supported).				
<b>Control 33</b> (Async only)	Sets second character in Prompt sequence (not supported).				
<b>Status 34</b> (Async only)	Returns the number of bits per character.				
<b>Control 34</b> (Async only)	<p>Sets the number of bits per character as follows:</p> <table> <tr> <td>0=5 bits/character</td> <td>2=7 bits/character</td> </tr> <tr> <td>1=6 bits/character</td> <td>3=8 bits/character</td> </tr> </table> <p><b>Reset Value</b> is determined by the Datacomm Configuration for HP-UX.</p>	0=5 bits/character	2=7 bits/character	1=6 bits/character	3=8 bits/character
0=5 bits/character	2=7 bits/character				
1=6 bits/character	3=8 bits/character				



- Status 35** Returns the number of stop bits per character.  
(Async only)
- Control 35** Sets the number of stop bits per character as follows:  
(Async only) 0=1 stop bit 2=2 stop bits
- Reset Value is determined by the Datacomm configuration for HP-UX.
- Status 36** Returns current Parity setting.
- Control 36** Sets the parity for transmitting and receiving asynchronous protocol.  
0=NONE; no parity bit is included with any characters.  
1=ODD; Parity bit SET if there is an EVEN number of  
"1"s in the character body.  
2=EVEN; Parity bit OFF if there is an ODD number of  
"1"s in the character body.
- Reset Value is determined by the Datacomm configuration for HP-UX.
- Status 37** Returns inter-character time gap in character times (not supported).  
(Async only)
- Control 37** Sets inter-character time gap in character times (not supported).  
(Async only)
- Status 38** Returns Transmit queue status (not supported).
- Status 39** Returns current Break time (not supported).  
(Async only)
- Control 39** Sets Break time in character times (not supported).  
(Async only)

# Table of Contents

---

## Chapter 15: The GPIO Interface

Introduction .....	15-1
Interface Description .....	15-2
Interface Configuration .....	15-4
Interface Select Code .....	15-5
Hardware Interrupt Priority .....	15-5
Data Logic Sense .....	15-5
Data Handshake Methods .....	15-5
Interface Reset .....	15-17
Outputs and Enters through the GPIO .....	15-18
ASCII and Internal Representations .....	15-18
Using a GPIO Interface in the HP-UX Environment .....	15-24
GPIO Timeouts .....	15-26
Using Alternate Data Representations .....	15-28
BCD Representation .....	15-28
Character Conversions .....	15-31
GPIO Interrupts .....	15-32
Types of Interrupt Events .....	15-32
Setting Up and Enabling Events .....	15-32
Interrupt Service Routines .....	15-34
Designing Your Own Transfers .....	15-37
Full Handshake Transfer .....	15-38
Interrupt Transfers .....	15-39
Using the Special-Purpose Lines .....	15-41
Driving the Control Output Lines .....	15-41
Interrogating the Status Input Lines .....	15-42
Using the PSTS Line .....	15-43
Summary of GPIO STATUS and CONTROL Registers .....	15-44
Summary of GPIO READIO and WRITEIO Registers .....	15-47
GPIO READIO Registers .....	15-47
GPIO WRITEIO Registers .....	15-49



# The GPIO Interface

---

## Introduction

This chapter should be used in conjunction with the *HP 98622A GPIO Interface Installation* manual. **The best way to use these two documents is to read this chapter before attempting to configure and connect the interface** according to the directions given in the installation manual. The reason for this order of use is that knowing how the interface works and how it is driven by BASIC programs will help you to decide how to connect it to your peripheral device.

The HP 98622 Interface is a very flexible parallel interface that allows you to communicate with a variety of devices. The interface sends and receives up to 16 bits of data with a choice of several handshake methods. External interrupt and user-definable signal lines are provided for additional flexibility. The interface is known as the General-Purpose Input/Output (GPIO) Interface for these reasons. This chapter describes the use of the interface's features from BASIC programs.

Use of some statements or suggestions for interfacing requires that you load the TRANS BIN file.



## Interface Description

The main function of any interface obviously to transfer data between the computer and a peripheral device. This section briefly describes the interface lines and how they function. Using the lines from BASIC programs is more fully described in subsequent sections.

The GPIO Interface provides **32 lines for data input and output**: 16 for input (DI0 — DI15), and 16 for output (DO0 — DO15).

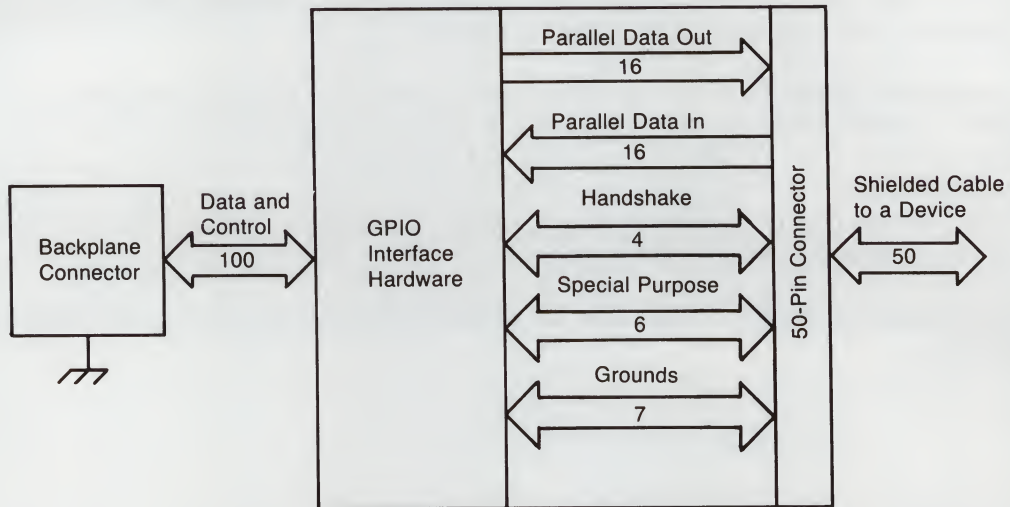


Figure 15-1. Block Diagram of the GPIO Interface

Two lines are dedicated to **handshaking** the data from source to destination device. The Peripheral Control line (PCTL) is controlled by the interface and is used to initiate data transfers. The Peripheral Flag line (PFLG) is controlled by the peripheral device and is used to signal the peripheral's readiness to continue the transfer process.

One line is used to signal External Interrupt Requests to the computer (EIR). The interface must be enabled to initiate interrupt branches for the interface to detect this request. The state of the line can also be read by the program.

Four general-purpose lines are available for any purpose that you may desire; two are controlled by the computer and sensed by the peripheral (CTL0 and CTL1), and two are controlled by the peripheral device and sensed by the computer (STI0 and STI1).

Both Logic Ground and Safety Ground are provided by the interface. Logic Ground provides the reference point for signals, and Safety Ground provides earth ground for cable shields.

---

## Interface Configuration

This section presents a brief summary of selecting the interface's configuration-switch settings. It is intended to be used as a checklist and to begin to acquaint you with programming the interface. **Refer to the installation manual for the exact location and setting of each switch.**

The following sample program checks a few of these switch settings on a GPIO Interface already installed in the computer and displays the settings. However, many of the settings cannot be determined from BASIC programs. If any of the displayed settings are different than desired, or if any settings are not already known, refer to the installation manual for switch locations and settings.

```
100  PRINTER IS 1    ! Select printer device.
110  PRINT CHR$(12) ! Clear screen.
120  !
130  DISP "Enter GPIO Interface Select Code (CONT=12)"
140  OUTPUT 2 USING "#,DD";12
150  ENTER 2;Isc
160  DISP
170  !
180  ASSIGN @Gpio TO Isc ! FORMAT ON default.
190  !
200  ! Read STATUS registers 0 and 1.
210  STATUS Isc;Card_id,Intr_stat
220  !
230  ! Is this card a GPIO?
240  IF Card_id<>3 THEN
250      PRINT "The interface at select code";Isc
260      PRINT "is not a GPIO Interface."
270      PRINT "Program stopped."
280      STOP
290  ELSE
300      PRINT "The card ID of the GPIO at"
310      PRINT "interface select code";Isc
320      PRINT "is";Card_id
330  END IF
340  PRINT
350  !
360  ! Calculate hardware interrupt priority.
370  Bits_5_and_4=BINAND(Intr_stat,32+16)
380  Switches=Bits_5_and_4 DIV 16
390  Hd_prior=Switches+3
400  PRINT "Hardware Interrupt Priority is";Hd_prior
410  PRINT
420  !
430  END
```



## Interface Select Code

In BASIC, allowable interface select codes range from 8 through 31; codes 1 through 7 are already used for built-in interfaces. The GPIO interface has a factory default setting of 12, which can be changed by re-configuring the “SEL CODE” switches on the interface.

## Hardware Interrupt Priority

Two switches are provided on the interface to allow selection of hardware interrupt priority. The switches allow hardware priority levels 3 through 6 to be selected. **Hardware priority** determines the order in which simultaneously occurring interrupt events are **logged**, while **software priority** determines the order in which interrupt events are **served** by the BASIC program.

## Data Logic Sense

The data lines of the interface are **normally low-true**; in other words, when the voltage of a data line is low, the corresponding data bit is interpreted to be a 1. This logic sense may be changed to high-true with the Option Select Switch. Setting the switch labeled “DIN” to the “0” position selects high-true logic sense of Data In lines. Conversely, setting the switch labeled “DOUT” to the “1” position inverts the logic sense of the Data Out lines. The default setting is “1” for both.

## Data Handshake Methods

This section describes the data handshake methods available with the GPIO Interface. A general description of the handshake modes and clock sources is given first. A more detailed discussion of each handshake is then given to allow you to choose the handshake mode, clock source, and handshake-line logic sense that is compatible with your peripheral device.

As a brief review, a data handshake is a method of synchronizing the transfer of data from the sending to the receiving device. In order to use any handshake method, **the computer and peripheral device must be in agreement as to how and when several events will occur**. With the GPIO Interface, the following events must take place to synchronize data transfers; the first two are optional.

- The computer may optionally be directed to perform a one-time “OK check” of the peripheral before beginning to transfer any data.
- The computer may also optionally check the peripheral to determine whether or not the peripheral is “ready” to transfer data.



- The computer must indicate the direction of transfer and then initiate the transfer.
- During OUTPUT operations, the peripheral must read the data sent from the computer while valid; similarly, the computer must clock the peripheral's data into the interface's Data In registers while valid during ENTER operations.
- The peripheral must acknowledge that it has received the data.

### **Handshake Lines**

**The GPIO handshakes data with three signal lines.** The Input/Output line, I/O, is driven by the computer and is used to signal the direction of data transfer. The Peripheral Control line, PCTL, is also driven by the computer and is used to initiate all data transfers. The Peripheral Flag line, PFLG, is driven by the peripheral and is used to acknowledge the computer's requests to transfer data.

### **Handshake Logic Sense**

Logic senses of the PCTL and PFLG lines are selected with switches of the same name. The logic sense of the I/O line is High for ENTER operations and Low for OUTPUT operations; this logic sense cannot be changed. The available choices of handshake logic sense and handshake modes allow nearly all types of peripheral handshakes to be accommodated by the GPIO Interface.

### **Handshake Modes**

There are two general handshake modes in which the PCTL and PFLG lines may be used to synchronize data transfers: Full-Mode and Pulse-Mode Handshakes. If the peripheral uses pulses to handshake data transfers **and** meets certain hardware timing requirements, the Pulse-Mode Handshake may be used. The Full-Mode Handshake should be used if the peripheral does not meet the Pulse-Mode timing requirements.

The handshake mode is selected by the position of the "HSHK" switch on the interface, as described in the installation manual. Both modes are more fully described in subsequent sections.

### Data-In Clock Source

Ensuring that the data are valid when read by the receiving device is slightly different for OUTPUT and ENTER operations. During OUTPUTs, the interface generally holds data valid while PCTL is in the Set state, so the peripheral must read the data during this period. During ENTERs, the data must be held valid by the peripheral until the peripheral signals that the data are valid (which clocks the data into interface Data In registers) or until the data is read by the computer. The point at which the data are valid is signalled by a transition of PFLG. The PFLG transition that is used to signal valid data is selected by the “CLK” switches on the interface. Subsequent diagrams and text further explain the choices.

### Optional Peripheral Status Check

Many peripheral devices are equipped with a line which is used to indicate the device's current “OK-or-Not-OK” status. If this line is connected to the Peripheral Status line (PSTS) of the GPIO Interface, and the computer may determine the status of the peripheral device by checking the state of PSTS. The logic sense of this line may be selected by setting the “PSTS” switch.

**If enabled**, the computer performs a **one-time check** of the Peripheral Status line (PSTS) **before initiating any transfers** as part of the data-transfer handshake. If PSTS indicates “Not OK,” Error 172 is reported; otherwise, the transfer proceeds normally. If this feature is not enabled, this one-time check is never made. This feature is available with both Full-Mode and Pulse-Mode Handshakes. See “Using the PSTS Line” for further details.

## Full-Mode Handshakes

The Full-Mode Handshake mode is described first for two reasons. The first reason is that the PCTL and PFLG transitions must always occur in the order shown, so only one sequence of peripheral handshake responses needs to be shown. Secondly, this mode will generally work when the Pulse-Mode Handshake may not be compatible with the peripheral's handshake signals. The Pulse-Mode Handshake is described in the next section.

The following diagrams show the order of events of the Full-Mode OUTPUT and ENTER Handshakes. These drawings are not drawn to any time scale; only the order of events is important. The I/O line has been omitted to simplify the diagrams; in all cases, it is driven Low before any OUTPUT is initiated by the computer and High before any ENTER is initiated.

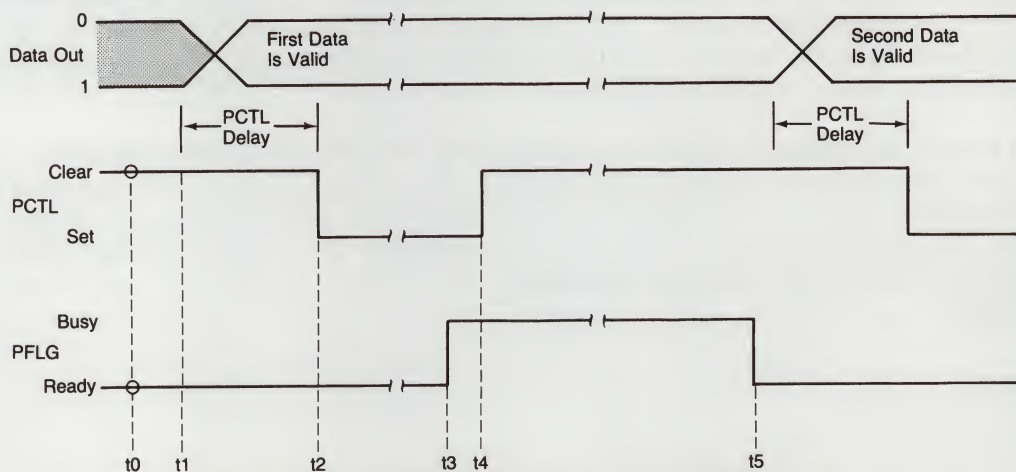


Figure 15-2. Diagram of Full-Mode OUTPUT Handshakes

With Full-Mode Handshakes, the computer first checks to see that the peripheral device is Ready before initiating the transfer of each byte/word (t0); with this handshake mode, the peripheral indicates **Ready when both PCTL is Clear and PFLG is Ready**. If the peripheral does not indicate Ready, the computer waits until a Ready is indicated.

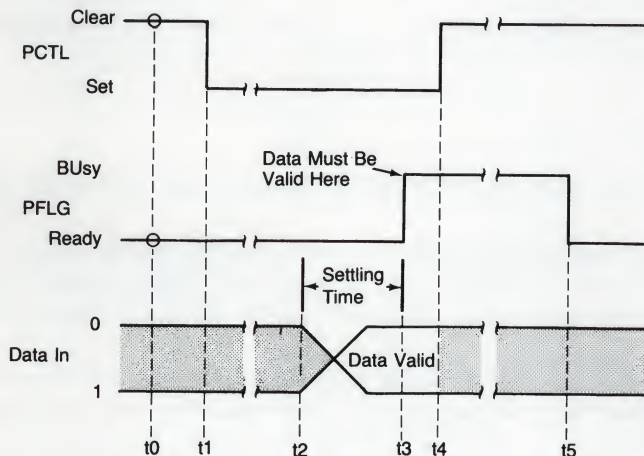
When a Ready is sensed, the computer places data on the Data Out lines (t1) and drives the I/O line Low (not shown). The interface then waits the PCTL Delay time before initiating the transfer by placing PCTL in the Set state (t2).



The peripheral acknowledges the computer's request by placing the PFLG line Busy (t3); this PFLG transition automatically Clears the PCTL line (t4). However, the computer cannot initiate further transfers until the peripheral is Ready with Full-Mode Handshake; the peripheral is not Ready until both PCTL is Clear and PFLG is Ready (t5).

The data on the Data Out lines is held valid from the time PCTL is Set until after the peripheral indicates Ready. The peripheral may read the data any time within this time period.

The PCTL and PFLG lines are used in the same manner in Full-Mode ENTER Handshakes as in Full-Mode OUTPUT Handshakes. However, there are three options available as to when the peripheral's data may be valid: at the Ready-to-Busy transition of PFLG (BSY clock source), at the Busy-to-Ready transition of PFLG (RDY clock source), and when the Data In lines are read with a STATUS statement (READ clock source). The first two of these options are shown in the following two diagrams; the READ clock source is discussed later in "Designing Your Own Transfers".

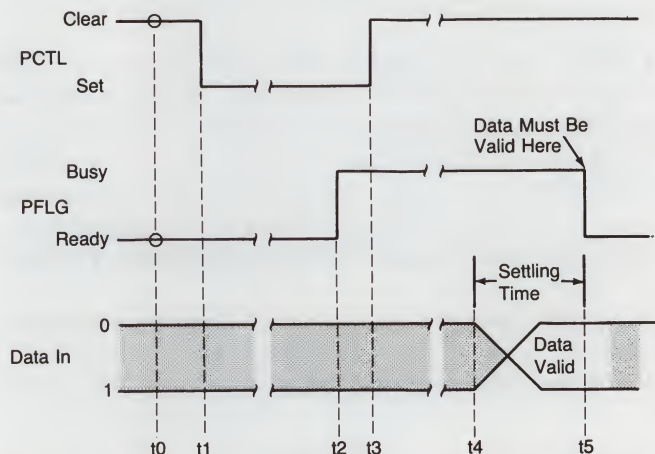


**Figure 15-3. Full-Mode ENTER Handshake with BSY Clock Source**

As with Full-Mode OUTPUT Handshakes, the computer first checks to see if the peripheral is Ready (t0); since PCTL is Clear and PFLG is Ready, the handshake may proceed. The computer places the I/O line in the High state (not shown) and then initiates the handshake by placing PCTL in the Set state (t1).



With the “BSY” clock source, the PFLG transition to the Busy state clocks the peripheral’s data into the interface’s Data-In registers; consequently, the peripheral must place data on the Data-In lines (t2), allowing enough time for the data to settle before placing PFLG in the Busy state (t3). This PFLG transition to the Busy state automatically Clears PCTL (t4). The next handshake may be initiated when PFLG is placed in the Ready state by the peripheral (t5).



**Figure 15-4. Full-Mode ENTER Handshake with RDY Clock Source**

As with other Full-Mode Handshakes, the computer first checks to see if the peripheral is ready (t0). Since PCTL is Clear and PFLG is Ready, the computer may drive the I/O line High (not shown) and initiate the handshake by placing PCTL in the Set state (t1).

The peripheral may acknowledge by placing PFLG Busy (t2), which automatically Clears PCTL (t3). Unlike the previous example, this transition does not clock data into the interface Data-In registers. With the “RDY” clock source, the peripheral must place the data on the Data-In lines (t4), allowing enough time for the data to settle before placing PFLG in the Ready state (t5). The computer may then initiate a subsequent transfer.

### Pulse-Mode Handshakes

The following drawings show the order of handshake-line events during Pulse-Mode Handshakes. Notice that the **main difference** between Full-Mode and Pulse-Mode Handshakes is that the **PFLG is not checked for Ready before the computer initiates Pulse-Mode Handshakes**; the computer may initiate a subsequent data transfer as soon as the PCTL line is Cleared by the Ready-to-Busy transition of PFLG.

Two cycles of data transfers are shown in these diagrams to illustrate that the computer need not wait for the PFLG=Ready indication with the Pulse-Mode Handshake. The first cycle shown in each diagram is a typical example of the first transfer of an I/O statement. The dashed PFLG line at the beginning of the second cycle shows that computer disregards whether or not PFLG is in the Ready state before the next transfer is initiated.

This absence of the PFLG check allows a **potentially higher data-transfer rate** than possible with the Full-Mode Handshake; however, in some cases, it also places additional timing restrictions on the peripheral's response time, as described in the text.

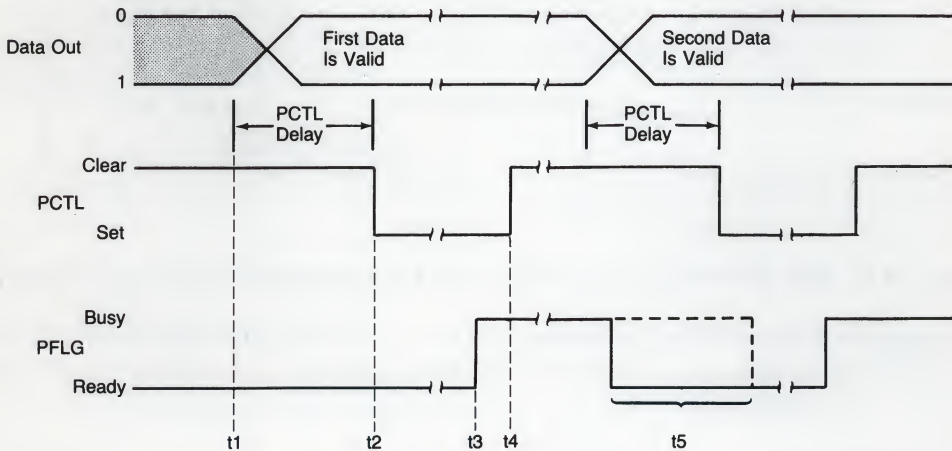
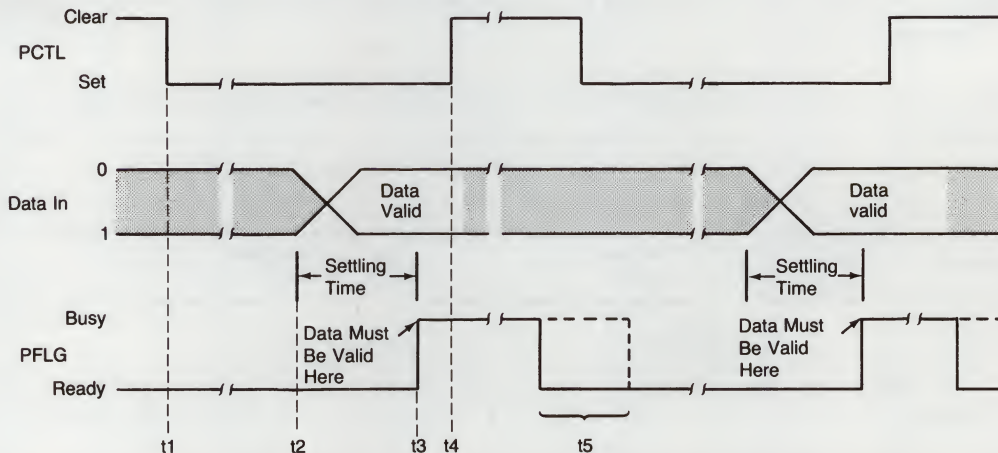


Figure 15-5. Busy Pulses with Pulse-Mode OUTPUT Handshake

The PFLG line is not checked for Ready before the computer drives the I/O line Low (not shown) and places data on the Data-Out lines ( $t_1$ ). A PCTL Delay time later, the interface initiates the transfer by placing PCTL in the Set state ( $t_2$ ).

The peripheral acknowledges by placing PFLG Busy (t3); this transition automatically Clears PCTL (t4). The dashed PFLG line shows that the computer may initiate another transfer any time after PCTL is Clear, possibly before the peripheral places PFLG in the Ready state (t5).

The Busy Pulse shown in the diagram is identical to the PFLG's response during the previous Full-Mode handshake; however, the Pulse-Mode Handshake works properly with this type of pulse **only** if the peripheral reads the data by the time PCTL is Clear (data should be read between t2 and t3). If the peripheral has not read the data by the time that PCTL is Clear, it might erroneously read the data for the second transfer, since the computer might have already changed the data and initiated the second transfer.

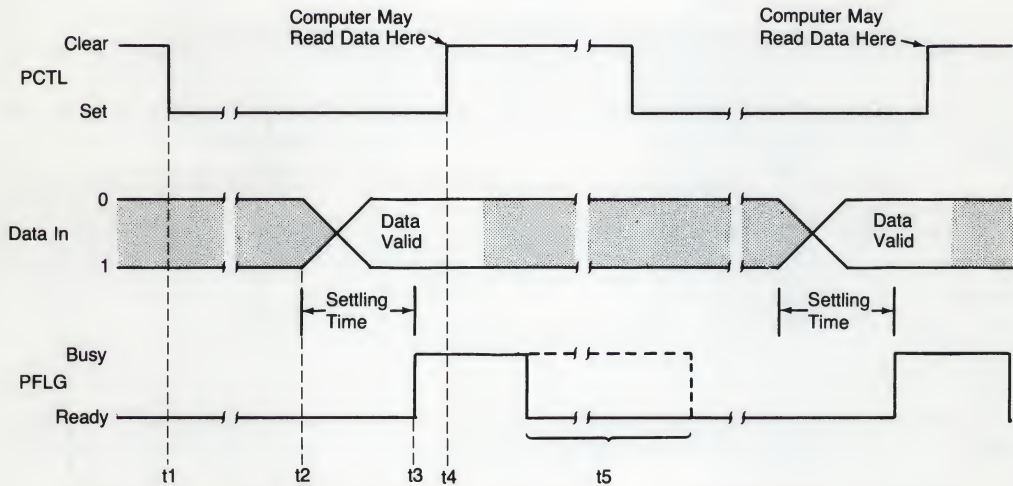


**Figure 15-6. Busy Pulses with Pulse-Mode ENTER Handshakes (BSY Clock Source)**

The computer does not have to check for PFLG to be Ready before placing I/O in the High state (not shown) and initiating the transfer by placing PCTL in the Set state (t1).



The peripheral must place data on the Data In lines (t2), allowing enough time for the data to settle before placing PFLG in the Busy state (t3). This Ready-to-Busy transition of PFLG automatically Clears PCTL. The dashed PFLG signal shows that the next transfer may be initiated before PFLG indicates Ready.



**Figure 15-7. Busy Pulses with Pulse-Mode ENTER Handshakes (RDY Clock Source)**

The computer does not have to check for PFLG to be Ready before placing I/O in the High state (not shown) and initiating the transfer by placing PCTL in the Set state (t1).

The peripheral must place data on the Data In lines (t2), allowing enough time for the data to settle before placing PFLG Busy (t3). This requirement **may seem contradictory**, since the clock source is the Busy-to-Ready transition of PFLG. However, with Pulse-Mode handshakes, the peripheral is assumed to be Ready whenever PCTL is Clear; consequently, the computer may read the data any time after PCTL is cleared by the Ready-to-Busy transition of PFLG. The PFLG transition to Busy Clears PCTL (t4), after which the peripheral may place PFLG Ready (t5).

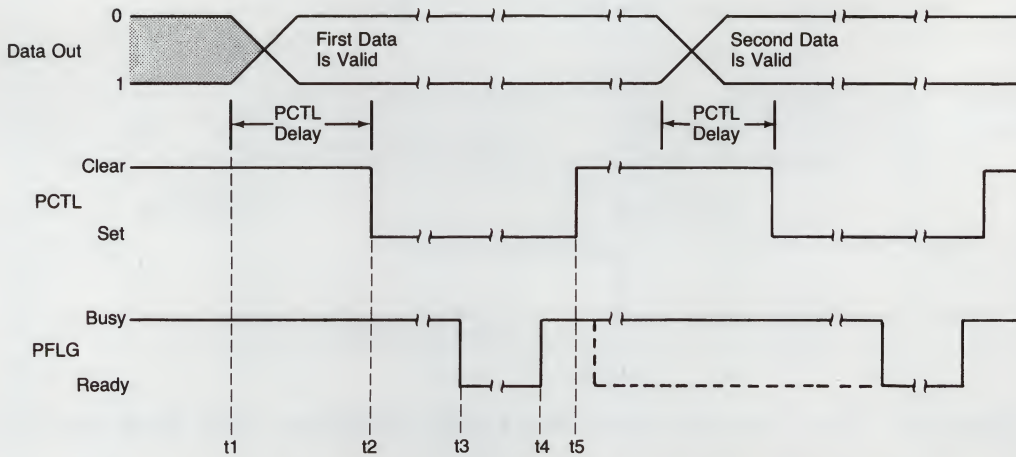


---

### Note

In order to use this type of pulse with the Pulse-Mode Handshake and RDY clock source, the peripheral must adhere to the stated timing restrictions.

---

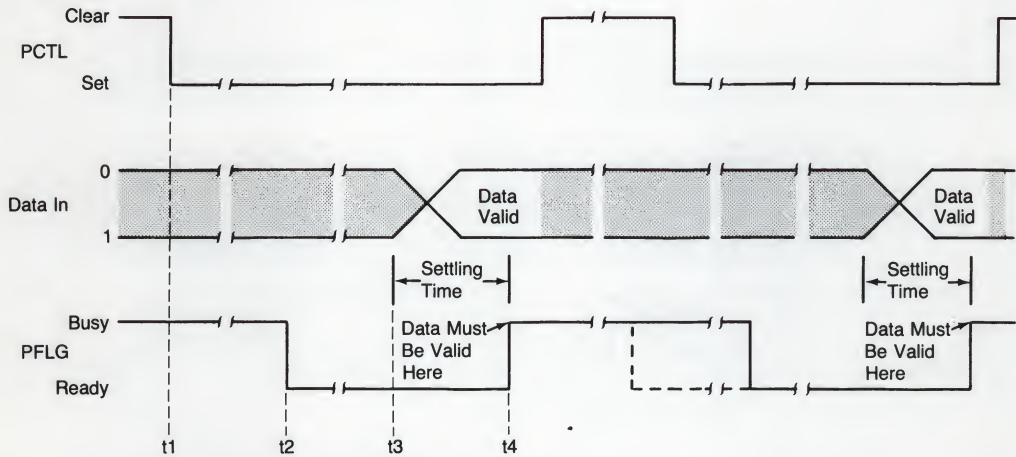


**Figure 15-8. Ready Pulses with Pulse-Mode OUTPUT Handshakes**

The PFLG line is not checked for Ready before the computer drives the I/O line Low (not shown) and places data on the Data Out lines (t1). At a PCTL Delay time later, the interface initiates the transfer by placing PCTL in the Set state (t2).

The peripheral later acknowledges by placing PFLG in the Ready state (t3). The handshake is completed by the peripheral placing PFLG in the Busy state (t4), which automatically Clears PCTL (t5).

If the peripheral uses the type of Ready pulses shown, either the Pulse-Mode handshake with default PFLG logic sense or Full-Mode handshake with inverted PFLG logic sense may be used. With this type of pulse, the data being output may be read by the peripheral as long as PCTL is Set.

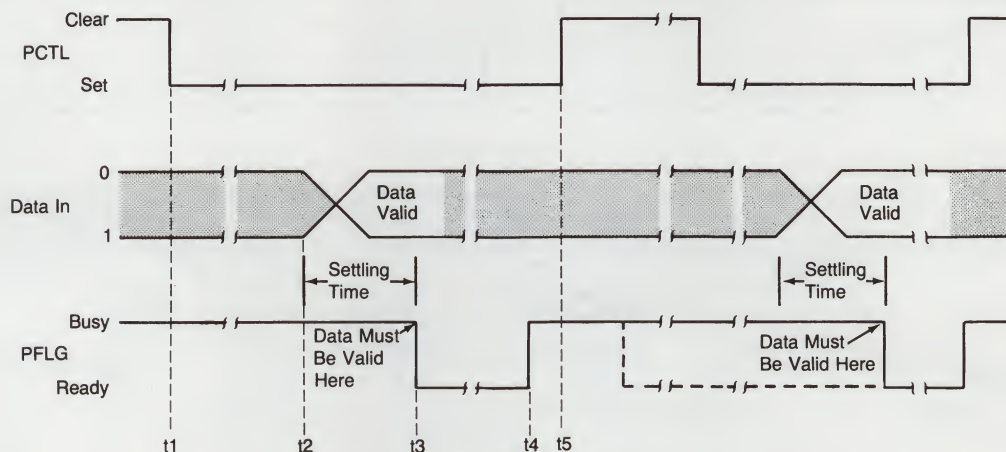


**Figure 15-9. Ready Pulses with Pulse-Mode ENTER Handshakes (BSY Clock Source)**

The computer does not have to check for PFLG to be Ready before placing I/O in the High state (not shown) and initiating the transfer by placing PCTL in the Set state ( $t_1$ ).

The peripheral acknowledges by placing PFLG in the Ready state ( $t_2$ ). The peripheral must place data on the Data In lines ( $t_3$ ), allowing enough time for the data to settle before placing PFLG in the Busy state ( $t_4$ ). With this type of pulse, events  $t_2$  and  $t_3$  may also occur in the reverse order.

The Ready-to-Busy transition of PFLG automatically Clears PCTL ( $t_4$ ). The dashed PFLG signal shows that the state of PFLG is not checked before the computer initiates a subsequent transfer.



**Figure 15-10. Ready Pulses w/ Pulse-Mode ENTER Handshakes (RDY Clock Source)**

The computer does not have to check for PFLG to be Ready before placing I/O in the High state (not shown) and initiating the transfer by placing PCTL in the Set state ( $t_1$ ).

The peripheral must place data on the Data In lines ( $t_2$ ), allowing enough time for the data to settle before placing PFLG Ready ( $t_3$ ). The peripheral places PFLG in the Busy state ( $t_4$ ), which automatically Clears PCTL ( $t_5$ ).

---

## Interface Reset

The interface should always be reset before use to ensure that it is in a known state. All interfaces are automatically reset by the computer at certain times: when the computer is powered on, when the **Shift-Reset** is pressed, and at other times described in the Reset Table<sup>1</sup>. The interface may be optionally reset at other times under control of BASIC programs. Two examples are as follows:

```
Gpio=12
CONTROL Gpio,0;1

Reset=1
CONTROL Gpio;Reset
```

The following action is invoked whenever the GPIO Interface is reset:

- The Peripheral Reset line (PRESET) is pulsed Low for at least 15 microseconds.
- The PCTL line is placed in the Clear state.
- If the DOUT CLEAR jumper is installed, the Data Out lines are all cleared (set to logic 0).
- The interrupt enable bit is cleared, disabling subsequent interrupts until re-enabled by the program.

The following lines are **unchanged** by a reset of the GPIO Interface:

- The CTL0 and CTL1 output lines.
- The I/O line.
- The Data Out lines, if the DOUT CLEAR jumper is not installed.

---

<sup>1</sup> The Reset Table is given in the Useful Tables of the *BASIC Language Reference*.



---

## Outputs and Enters through the GPIO

This section describes techniques for outputting and entering data through the GPIO Interface. The mechanism by which data are communicated are the electrical signals on the data lines. The actual signals that appear on the data lines depend on three things:

- the data currently being transferred,
- how this data is being represented,
- the logic sense of the data lines.

Brief explanations of ASCII and internal data representation are given in the “Interfacing Concepts” chapter. Complete details of the freefield convention and effects of IMAGE specifiers during OUTPUT and ENTER statements are described in the “Outputting Data” and “Entering Data” chapters, respectively. The section of the chapter “I/O Path Attributes” called “The FORMAT OFF Attribute” describes how internal-form data are represented during OUTPUT and ENTER. This section gives simple examples of how several representations are implemented during OUTPUTs and ENTERs through the GPIO Interface.

### ASCII and Internal Representations

When data are moved through the GPIO Interface, the **data are generally sent one byte at a time, with the most significant byte first**. This byte-mode transfer is independent of whether FORMAT ON or FORMAT OFF is the I/O path attribute. However, there are **two exceptions**; data are represented by words when the “W” image specifier is used and when numeric data are moved with reads of STATUS register 3 and writes to CONTROL register 3. The following diagrams illustrate which data lines are used during byte and word transfers.

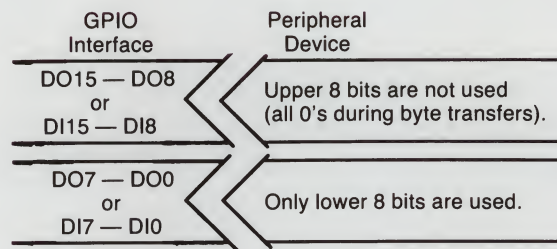


Figure 15-11. Diagram of Byte Transfers

### Example Statements that Output Data Bytes

The following diagrams show the actual logic signals that appear on the least significant data byte (DO7 thru DO0) as the result of the corresponding OUTPUT statement; the most significant byte is always zeros with byte transfers. The actual logic levels depend on how the data lines are configured (i.e., as Low-true or High-true).

```
ASSIGN @Gpio TO 12
OUTPUT @Gpio;"ASCII"
```

Signal Line							ASCII
DO7	.....	DO0					Char.
0	1	0	0	0	0	1	A
0	1	0	1	0	0	1	S
0	1	0	0	0	0	1	C
0	1	0	0	1	0	0	I
0	1	0	0	1	0	0	I
0	0	0	0	1	1	0	C <sub>R</sub>
0	0	0	0	1	0	1	L <sub>F</sub>

```
Gpio=12
Number=-4
OUTPUT Gpio USING "MD.DD";Number
```

Signal Line							ASCII
DO7	.....	DO0					Char.
0	0	1	0	1	1	0	-
0	1	1	0	0	1	0	4
0	0	1	0	1	1	1	,
0	0	1	1	0	0	0	Q
0	0	1	1	0	0	0	Q
0	0	0	0	1	1	0	C <sub>R</sub>
0	0	0	0	1	0	1	L <sub>F</sub>

```
ASSIGN @Gpio TO 12;FORMAT OFF
Integer_1=256*85+76
OUTPUT @Gpio;Integer_1
```

Signal Line							ASCII
DO7	.....	DO0					Char.
0	1	0	1	0	1	0	U
0	1	0	0	1	1	0	L

```

ASSIGN @Gpio TO 12;FORMAT OFF
String$="1234"
OUTPUT @Gpio;String$

```

Signal Line								ASCII
DO7	.....	DO0						Char.
0	0	0	0	0	0	0	0	N <sub>u</sub>
0	0	0	0	0	0	0	0	N <sub>u</sub>
0	0	0	0	0	0	0	0	N <sub>u</sub>
0	0	0	0	0	1	0	0	E <sub>t</sub>
0	0	1	1	0	0	0	1	1
0	0	1	1	0	0	1	0	2
0	0	1	1	0	0	1	1	3
0	0	1	1	0	1	0	0	4

### Example Statements that Enter Data Bytes

The following diagrams show the variable values that result from the logic signals being present during the corresponding ENTER statement on the least significant data byte (DI7 thru DI0); the most significant byte is always ignored with byte transfers. The actual logic levels required depend on how the data lines are configured (i.e., as Low-true or High-true).

```

ENTER @Gpio USING "#,B";Byte
DISP "Value entered=";Byte

```

Value entered= 65

Signal Line								ASCII
DI7	.....	DI0						Char.
0	1	0	0	0	0	0	1	A

```

ENTER 12;String$
DISP "String entered= ";String$

```

String entered= ruok?

Signal Line								ASCII
DI7	.....	DI0						Char.
0	1	1	1	0	0	1	0	r
0	1	1	1	0	1	0	1	u
0	1	1	0	1	1	1	1	o
0	1	1	0	1	0	1	1	k
0	0	1	1	1	1	1	1	?
0	0	0	0	1	0	1	0	L <sub>F</sub>

```

REAL Number
ASSIGN @Gpio TO 12
ENTER @Gpio;Number
DISP "Number=";Number

```

Number= 2

Signal Line								ASCII
DI7	.....	DI0						Char.
0	1	0	0	0	0	0	0	Ⓢ
0	0	0	0	0	0	0	0	N <sub>U</sub>
0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	,
0	0	0	0	0	0	0	0	,
0	0	0	0	0	0	0	0	,
0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	N <sub>U</sub>

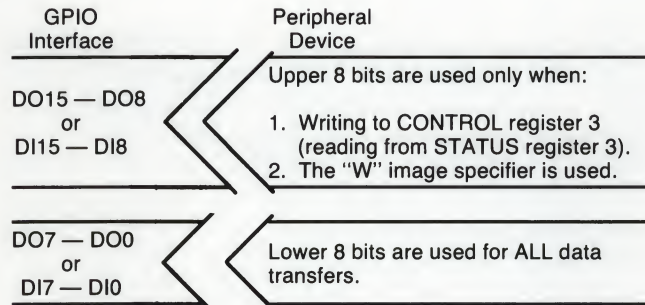


Figure 15-12. Diagram of Word Transfers



### Example Statements that Output Data Words<sup>1</sup>

The following diagrams show the logic signals that appear on the Data Out lines as a result of the corresponding BASIC statements and numeric values. All numeric values are first rounded to an INTEGER value before being placed on the Data Out lines. The actual logic level that appears on each line depends on how the lines have been configured (i.e., as High-true or Low-true).

```
Word=3*256+3
OUTPUT @Gpio USING "#,W";Output_word
```

Signal Lines															
DO15 .....								DO7 ..... DO0							
0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1

```
Output_16_bits=-1
CONTROL Gp_isc,3;Output_16_bits
```

Signal Lines															
DO15 .....								DO7 ..... DO0							
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

It is important to note that *no output handshake is executed when the CONTROL statement is executed*; only the states of the Data Out lines and the I/O line are affected. Handshake sequence, if desired, must be performed by BASIC statements in the program. See “Designing Your Own Transfers” for design suggestions.

<sup>1</sup> Data are automatically sent as words when using an I/O path with the WORD attribute. See the “I/O Path Attributes” chapter for further information.

### Example Statements that Enter Data Words<sup>1</sup>

The following diagrams show the variable values that result from entering the logic signals on the Data In lines. Note that all sixteen-bit values entered are interpreted as INTEGER values.

Signal Lines															
DI15 .....								DI7 ..... DI0							
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1

```
ENTER 12 USING "#,W";Enter_16_bits
DISP "INTEGER entered=";Enter_16_bits
```

```
INTEGER entered= 511
```

Signal Lines															
DI15 .....								DI7 ..... DI0							
1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0

```
STATUS Gp_isc,3;Enter_16_bits
DISP "INTEGER entered=";Enter_16_bits
```

```
INTEGER entered= -512
```

It is important to note that no enter handshake is performed when the STATUS statement is executed. The only actions taken are the I/O line being placed in the High state and the Data In registers being read. If an enter handshake is required, it must be performed by the BASIC program. See "Designing Your Own Transfers" for design suggestions.

Remember also that the Data In Clock source is solely determined by the switch setting on the interface card. Thus, when the STATUS statement is used to read the Data In lines, the data on the lines may or may not be clocked into the registers when the statement is executed. If the data are to be clocked in by the STATUS statement, the "READ" clock source must be selected. See the installation manual for further details.

<sup>1</sup> Data are automatically received as words when using an I/O path with the WORD attribute. See the "I/O Path Attributes" chapter for further information.

## Using a GPIO Interface in the HP-UX Environment

This section explains the interface locking and burst I/O, which are useful when using an interface in the HP-UX environment.

### Locking an Interface to a Process

In a multi-user environment, interface cards are usually accessible to several users. BASIC/UX supports this sharing by making no attempt to guarantee exclusive access to an interface *unless it is directed to do so*. This allows you to access instruments, for instance, on an GPIO bus that is shared with other peripherals. Although this is not a recommended configuration, it is allowed.

BASIC/UX provides interface locking to support exclusive access to an interface. When an interface is locked to a process, all other processes are prevented from using that interface. For instance, this feature can prevent the loss of important data while a process is taking measurements from an instrument by keeping other users or processes from using the same interface.

Interface locking is enabled and disabled by using pseudo-register 255 and the interface's select code. For example:

`CONTROL 12,255;1`      *Enables GPIO interface locking.*

`CONTROL 12,255;0`      *Disables GPIO interface locking.*

In order to be a “good citizen” on a multi-user system, you should unlock an interface after you no longer need to have it locked.

Note that attempting to lock an GPIO connected to a system disc will result in an error.

In addition, attempting to lock an interface that is already locked to another process will cause a program to suspend execution until:

- The interface is unlocked (by the other process to which it is currently locked).
- A timeout occurs.
- You press `Reset` or `Clr I/O`.



### Using the Burst I/O Mode

The default mode of HP-UX I/O transactions requires many time consuming HP-UX system calls to send data to the destination.

Another method, “burst I/O”, maps the interface into your “user address space”, thereby bypassing the memory buffer. This direct-write method decreases the number of calls to HP-UX I/O system routines, which establishes a short, highly tuned path for performing I/O operations. The interface is also implicitly locked when burst mode is enabled (see above explanation of interface locking).

Burst I/O provides the fastest I/O performance available with BASIC/UX for the “smaller” I/O transactions that are typical of many instruments. For instance, an 8-byte ENTER operation is over an order of magnitude faster when burst mode is enabled. For larger I/O operations, of more than 4 000 bytes for example, burst mode becomes increasingly slower than the default (buffered or DMA) I/O modes.

Burst I/O is enabled and disabled by using register 255 and the interface’s select code. For example:

CONTROL 12,255;3      *Enables GPIO interface burst I/O.*

CONTROL 12,255;0      *Disables GPIO interface burst I/O.*

In order to be a “good citizen” on a multi-user system, you should unlock an interface after you no longer need to have it locked.

In addition, attempting to use burst mode with an interface that is already locked to another process will cause a program to suspend execution until:

- The interface is unlocked (by the other process to which it is currently locked).
- A timeout occurs.
- You press **Reset** or **Clr I/O**.

Note also that you cannot set up an ON TIMEOUT for an interface when using burst mode.



## GPIO Timeouts

Timeout events were generally discussed in the chapter “Interface Events”. However, specific details of the affects of the time parameter on the event’s occurrence were not described. This section explains how the time parameter is measured and describes typical service routines.

### Timeout Time Parameter

There are two general time intervals measured and compared to the specified TIMEOUT time. The first interval is measured between the computer initiating the first handshake (PCTL=Set) and the peripheral signalling Ready (with the PFLG line). If the peripheral does not indicate readiness<sup>1</sup> by the specified TIMEOUT time parameter, a TIMEOUT event occurs.

The time elapsed during each handshake is also measured and compared to the TIMEOUT time. The timing begins when the transfer is initiated (PCTL Set by the computer) and, in general, ends when the peripheral responds on the PFLG line.

Keep in mind that the TIMEOUT time parameter specifies the **minimum** time that the computer will wait before initiating the ON TIMEOUT branch. However, the computer may occasionally wait an additional 25% of the specified time parameter before initiating the branch. For instance, if a time of 0.4 seconds is specified, the computer will wait at least 0.4 seconds for the handshake to be completed, but it may occasionally wait up to 0.5 seconds before initiating the ON TIMEOUT branch.

Note that timeouts do not occur when burst mode is enabled.

---

<sup>1</sup> The computer optionally reads the state of the PSTS line before initiating the transfer. See “Using the PSTS Line” for further details.

## Timeout Service Routines

The service routine usually responds by determining if the peripheral is functioning properly ("ok") or is down ("not ok"). The simplest action that might be taken by the computer is to read the state of the PSTS signal line, as shown in the following service routine.

```
100  Gpio=12
110  ON TIMEOUT Gpio,.08 GOSUB Gpio_down
    .
    .
200  OUTPUT Gpio;String$
210  ! Next line.
    .
    .
300  Gpio_down: STATUS Gpio,5;Periph_status
310          Psts=BIT(Periph_status,3) ! Read PSTS.
320          IF NOT Psts THEN
330              PRINT "GPIO interface is "
340              PRINT "non-functional"
350              PRINT "Program paused."
360              PAUSE
370          ELSE
380              ! Take other action.
390          END IF
400      RETURN
```

A TIMEOUT has been set up to occur if the peripheral takes approximately more than .08 second to complete its response during a data transfer; how the peripheral completes its response depends on the handshake mode currently selected. With Pulse-Mode Handshakes, the peripheral completes its response by using PFLG to Clear PCTL; with Full-Mode Handshakes, the response is complete **only** after PCTL has been Cleared **and** PFLG is in the Ready state.

**When a TIMEOUT occurs, the computer automatically executes an Interface Reset;** the PCTL line is Set and then Cleared, and the PRESET line is pulsed Low. See the section called "Interface Reset" for further effects. The Service routine checks the PSTS line to see if the peripheral is OK or not OK. If not OK, a message is displayed and the program is paused; if OK, program execution is returned to the line following that in which the TIMEOUT occurred. The service routine may be programmed to attempt the transfer again, if desired; however, the automatic Reset performed when the TIMEOUT occurred may make this type of response difficult to implement.

---

## Using Alternate Data Representations

As with any other interface, representations other than the ASCII or internal representations may sometimes be more meaningful to the peripheral. This section briefly describes a few techniques for implementing alternate data representations.

### BCD Representation

With OUTPUT and ENTER statements, numeric values are either represented by ASCII characters or by one of the internal representations (INTEGER or REAL). Another common method of representing numeric data is to use four-bit, binary-coded decimal (BCD) characters. Only ten of the available sixteen bit patterns need to be used to represent decimal digits "0" through "9". The remaining six patterns can be used for sign, decimal point, exponent, and other special characters, as required by the application.

The following bit patterns have been chosen arbitrarily to correspond to numeric characters<sup>1</sup>. Note that this representation cannot be used if more than six other characters are to be represented.

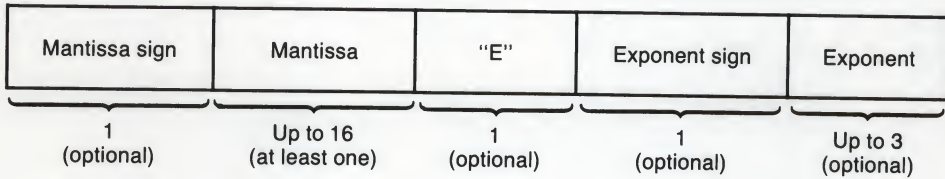
---

<sup>1</sup> This is also the data representation used by the HP 98623 BCD Interface. See the "BCD Interface" chapter for further information.

Table 15-1. Bit Patterns for Numeric Characters

Decimal Digit	Bit Pattern				Other Character	Bit Pattern			
	MSB		LSB			MSB		LSB	
0	0	0	0	0	Line-Feed	1	0	1	0
1	0	0	0	1		+	1	0	1
2	0	0	1	0	,	1	1	0	0
3	0	0	1	1	—	1	1	0	1
4	0	1	0	0	E	1	1	1	0
5	0	1	0	1	.	1	1	1	1
6	0	1	1	0					
7	0	1	1	1					
8	1	0	0	0					
9	1	0	0	1					

The following subprogram assumes that BCD numbers are to be entered through the GPIO Interface. Sixteen BCD characters are represented by four 16-bit words from the peripheral. The sixteen four-bit BCD characters have the following general format.





Each BCD character is represented by four bits of data. The first word entered contains the four most significant BCD characters, and the last word contains the least significant. The program changes the BCD characters to their ASCII representation and then uses the number builder to generate the corresponding numeric value.

```

100  ASSIGN @Gpio TO 12
110  !
120  ! Define conversion string.
130  Conv$="0123456789"&CHR$(10)&"+" ,-E."
140  !
150  CALL Enter_bcd(@Gpio,Conv$,Number)
160  OUTPUT 1;"The BCD number is ";Number
170  !
180  END
190  !
200  !
210  SUB Enter_bcd(@Device,Conv$,Number)
220  COM /Enter_bcd/ INTEGER Word(1:4)
230  !
240  ! Enter 4 words (=16 BCD digits).
250  ENTER @Device USING "#,W";Word(*)
260  !
270  FOR W=1 TO 4 ! Process four words.
280  !
290  ! Shift right multiples of four bits.
300  FOR Bits_rt=12 TO 0 STEP -4
310  Shifted_word=SHIFT(Word(W),Bits_rt)
320  Four_lsb=BINAND(Shifted_word,15) ! Mask MSB's.
330  Ascii_char$=Conv$[Four_lsb+1;1] ! LSB's = index.
340  Number$=Number$&Ascii_char$
350  NEXT Bits_rt
360  !
370  NEXT W
380  !
390  ENTER Number$;Number ! Use number builder.
400  SUBEND ! Returns BCD number as "Number".

```

## Character Conversions

One of the most common needs of a computer is to convert<sup>1</sup> certain unused or disallowed bit patterns into meaningful or allowed bit patterns. A typical example is to change the radix character from a decimal point to a comma. For instance, the following ASCII characters represent the same number.

U.S. Representation	European Representation
1,234,567.89	1.234.567,89

A remedy is needed to allow these types of numbers to be entered through the number builder. To enter a number with the preceding European format, the commas must be changed to periods and the periods changed to spaces. The following routine changes the numeric radix from the European to the US convention when numeric data are entered through the GPIO.

```
100  ! Generate string with no conversions.
110  DIM Conv$(256)
120  FOR Code=0 TO 255
130    Conv$(Code+1)=CHR$(Code)
140  NEXT Code
150  !
160  ! Then define the conversions.
170  Conv$(NUM("."))+1;1)=" " ! Change "." to " "
180  Conv$(NUM(",")+1;1)=". " ! Change "," to ". "
190  !
200  !
210  Number$="123.456,789"
220  PRINT "Before conversion ";Number$
230  CALL Convert(Conv$,Number$)
240  PRINT "After conversion ";Number$
250  !
260  END
270  !
280  !
290  SUB Convert(Conv$,Data$)
300  !
310  FOR Char_pos=1 TO LEN(Data$)
320    Index=NUM(Data$(Char_pos))+1
330    Data$(Char_pos;1)=Conv$(Index;1)
340  NEXT Char_pos
350  !
360  ! Returns Data$ with converted characters.
370  SUBEND
```

<sup>1</sup> Conversions can also be made by using the CONVERT attribute. See the "I/O Path Attributes" chapter for further information.

If more characters are to be converted, simply change the default (standard ASCII) character in Conv\$ to the desired code. The speed of the conversion is not affected by the number of characters to be converted. This routine works for either input or output, but the characters to be converted must be in a string variable.

---

## GPIO Interrupts

This section describes the types of and techniques for using the interrupts available on the GPIO Interface.

### Types of Interrupt Events

The GPIO Interface can sense **two interrupt events**: the first is the interface becoming “Ready” for subsequent handshakes, and the second is the External Interrupt Request line (EIR) being driven to logic low by the peripheral. As with all interfaces, both events initiate identical computer responses — the service routine must be able to determine which of these interrupts have occurred if both are enabled to initiate interrupts.

Both of these types of interrupt events are **level-sensitive**; in other words, the signal that caused the event should be maintained until the program has time to determine which event has caused the interrupt. Further explanation follows in this section.

### Setting Up and Enabling Events

When either event occurs, the interrupt is logged by the BASIC operating system. After logging the occurrence, any further interrupts from the GPIO Interface are automatically disabled until specifically enabled by a program. All further computer responses to either event depend entirely on the BASIC program currently in memory.

The following program segment shows the steps involved in setting up and enabling Ready interrupts.

```
100  Gpio=12
110  ON INTR Gpio GOSUB Gpio_serv
120  !
130  Mask=2
140  ENABLE INTR Gpio;Mask
```



The value of the interrupt mask determines which, if any, of the GPIO interrupt events are to be enabled to initiate the corresponding branch. Bits of the Interrupt Mask register have the following definitions.

### Interrupt Enable Register (ENABLED INTR)

Most Significant Bit

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used						Enable Interface Ready Interrupts	Enable EIR Interrupts
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**Interface Ready** — Setting this bit (1) enables an interrupt to initiate the ON INTR branch when the interface detects that it is Ready to handshake data. If Full-Mode Handshake is selected (with the Option Select switch), the Ready event is PCTL=Clear and PFLG=Ready. With Pulse-Mode Handshake, the event is PCTL=Clear (independent of the state of PFLG).

**External Interrupt Request** — Setting this bit (1) enables an interrupt to initiate the ON INTR branch when the interface senses an External Interrupt Request (EIR line=Low).



## Interrupt Service Routines

If both events are enabled, the service routine must be able to differentiate between the two. And, if both have occurred, the service routine must be able to service both causes. The following registers contain the current state of the Interface Ready flag and EIR signal lines, from which the interrupt cause(s) may be determined.

**STATUS Register 4**      Interface is ready for a subsequent data transfer; 1=Ready, 0=Busy.

**STATUS Register 5**      Peripheral Status

**Most Significant Bit**

**Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	PSTS Ok	EIR Line Low	STI1 Line Low	STI0 Line Low
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

As mentioned in preceding paragraphs, these two interrupt causes are both **level-sensitive** events, not edge-triggered events. This fact has **two important implications**. The **first** is that, for an event to be recognized, the corresponding signal line must be held in the interrupting state until the computer can interrogate the line's logic state. If the signal line's state is changed before the service routine checks the line, the interrupt may be "missed". This will happen only if both events are enabled; if only one event is enabled, determining the cause may not be necessary.

The **second implication** is that the service routine must be able to acknowledge the request in order for the peripheral device to remove the request. If the request is not removed after service, the same request may be serviced more than once.

The following program shows a simple example of servicing an External Interrupt Request. Note that only EIR-type interrupts have been enabled and that the peripheral device provides its own interrupt cause with signals on the STIO and STI1 lines.

```
100  PRINTER IS 1
110  Gpio=12
120  CONTROL Gpio;1 ! Reset Interface.
130  !
140  ON INTR Gpio GOSUB Gpio_serv
150  ENABLE INTR Gpio;1 ! Enable EIR-type only.
160  !
170  ! Show concurrent processing.
180 Loop: Counter=Counter+1
190      DISP Counter
200      GOTO Loop
210  !
220  STOP
230  !
240 Gpio_serv: !
250  STATUS Gpio,5;Periph_status ! Check EIR line.
260  IF BIT(Periph_status,2) THEN ! EIR interrupt.
270  !
280      IF BIT(Periph_status,0) THEN ! STIO=True.
290          BEEP
300          PRINT "Improper value; type in correct"
310          PRINT "value, and press ENTER."
320          PRINT
330          ENTER 2;Value
340          OUTPUT Gpio;Value
350      END IF
```

```

360      !
370      IF BIT(Periph_status,1) THEN ! STI1=True.
380          BEEP
390          PRINT "Reading of: ";Reading;" out of range"
400          PRINT "No other action will be taken."
410          PRINT
420          WAIT 2
430          BEEP
440      END IF
450      !
460  END IF
470      !
480      ! Put Ready service routine here.
490      !
500      !
510  ENABLE INTR Gpio          ! Use same mask.
520  RETURN
530      !
540  END

```

A slightly different method that peripherals use to communicate the cause of their interrupt request is to place the interrupt cause on the data lines concurrent with the interrupt request. The service routine can determine the cause by reading STATUS register 3 and take the appropriate action.

Notice that the service routine indicates a likely place for a Ready-interrupt service routine. The Service routine must check for the Ready condition, acknowledge the interrupt, and then take the desired action. In this case, no service action has been defined because Ready interrupts have not been enabled. The next section provides an example of a Ready interrupt service routine.

---

## Designing Your Own Transfers

Other specialized methods of handshaking data can be designed according to your specific needs. The methods of synchronizing data transfers are as flexible as the GPIO Interface hardware. However, the general techniques will probably still require the fundamental handshake features: initiation by the sending device, acknowledgement from the receiving device, and agreement as to when the data are valid. The TRANSFER statement can be used to transfer data. See the chapter “Advanced Transfer Techniques” for further information.

A wide choice of initiating events is available; obvious possibilities include use of the PCTL, EIR, or CTL0 (or CTL1) lines to signal the start of the transfer. Data can be placed on the Data Out lines by writing to CONTROL register 3, or data can be clocked into the Data In registers by reading STATUS register 3. Sensing acknowledgement from the peripheral can be accomplished by reading the state of such lines as PFLG, PSTS, EIR, or STI0 (or STI1).

The feature common to all of these methods is that each byte (or word) of data must be transferred individually. If an entire block of data is to be entered or output, the BASIC program that implements the transfer must keep a “pointer” to which byte/word is to be transferred.



## Full Handshake Transfer

The following program implements a handshake similar to the Full OUTPUT Handshake by controlling the PCTL and sensing the PFLG and PCTL lines. The actual "Output" routine consists of lines 150 through 190. Timeout capability can easily be included in the routine, if so desired.

```
100 DATA 65,66,67,68,69
110 !
120 STATUS 12,5;Periph_status ! Check PSTS.
130 IF BIT(Periph_status,3) THEN ! PSTS True.
140 !
150 FOR Char=1 TO 5
160 READ Code
170 Wait: STATUS 12,4;Interface_ready
180 IF NOT Interface_ready THEN Wait
190 Output: CONTROL 12,3;Code ! Data onto lines.
200 CONTROL 12,1;1 ! Set PCTL.
210 NEXT Char
220 !
230 ELSE ! PSTS False.
240 PRINT "Peripheral error"
250 PAUSE
260 END IF
270 !
280 END
```

Notice that each byte of data must be output separately and that the program must keep track of which byte, of several, is to be sent. Keep in mind that the data written to CONTROL register 3 is **16-bit words**; in this case, the most significant eight bits (byte) is all zeros. Also, using FOR...NEXT loops to index each byte/word to be sent may not be the most expedient way of sending data, so your particular application may use alternative methods for handling the data.

The following subprogram implements a handshake similar to the Full ENTER handshake.

```
170 SUB Enter_word(@Device,Data_word)
180 !
190 Wait1: STATUS 12,4;Interface_ready
200     IF NOT Interface_ready THEN Wait1
210     STATUS 12,3;Dummy_read ! I/O High.
220     CONTROL 12,1;1 ! Set PCTL.
230 Wait2: STATUS 12,4;Interface_ready
240     IF NOT Interface_ready THEN Wait2
250     STATUS 12,3;Data_word ! Enter word.
260     !
270 SUBEND
```

The appropriate Data-In Clock source should be selected to ensure the data are clocked into the registers when valid. Refer to the installation manual for further details.

## Interrupt Transfers

The interrupt capabilities of the GPIO Interface can be used to synchronize the transfer of data between the computer and peripheral. These examples describe simple methods of synchronizing the transfer of data by using both the EIR and the PFLG line. See the section of this chapter called “GPIO Interrupts” for further explanation of GPIO interrupts in general.

General interrupt transfers through the GPIO Interface involve the following elements:

- placing data on (or reading data from) the data lines
- signaling to the peripheral device to initiate the transfer
- continuing processing until an interrupt is received, at which time the handshake is finished and transfer of the next byte/word can be initiated.

Examples of using Ready interrupts to implement interrupt transfers are given in the remainder of this section.

## Ready Interrupt Transfers

The Ready interrupt event occurs when the GPIO Interface becomes “Ready”. Whether or not the GPIO Interface is Ready depends on the currently selected handshake mode. If Full-Mode Handshake is selected, the interface is Ready if **both** the PFLG line is Ready and the PCTL line is Clear; if Pulse-Mode is selected, the interface is Ready if PCTL is in the Clear state, regardless of the state of PFLG. The following program shows how to implement Ready interrupt transfers.

```

100  PRINTER IS 1
110  Gpio=12
120  CONTROL Gpio;1 ! Reset Interface.
130  ON INTR Gpio GOSUB Ready_xfer
140  !
150  DIM Data_out$[256]
160  Data_out$="123ABC"
170  Pointer=1
180  Size=LEN(Data_out$)
190  !
200  ! Initiate the transfer.
210  GOSUB Ready_xfer
220  !
230  ! Show concurrent processing.
240 Loop: Counter=Counter+1
250      DISP Counter
260      GOTO Loop
270  !
280  STOP
290  !
300  ! The branch to this subroutine is initiated
310  ! first by the program, but thereafter by
320  ! Ready Interrupt events.
330  !
340 Ready_xfer: !
350      !
360  IF Pointer<=Size THEN
370      Byte_out=NUM(Data_out$[Pointer;1])
380      PRINT Data_out$[Pointer;1];" sent"
390      CONTROL Gpio,3;Byte_out ! Place data on lines.
400      Pointer=Pointer+1
410      CONTROL Gpio,1;1          ! Set PCTL.
420      ENABLE INTR Gpio;2        ! Enable Ready INTR's.
430      RETURN
440      !
450  ELSE
460      DISABLE INTR Gpio          ! Disable after done.
470      RETURN
480      !
490  END IF
500  !
510  !
520  END

```

Interrupt transfers that use the EIR line are similar to Ready interrupt transfers. The main difference is that the interrupt-initiating event is the EIR line, rather than the PCTL line (and PFLG if in Full Handshake mode) indicating Interface Ready.

## Using the Special-Purpose Lines

Four special-purpose signal lines are available for a variety of uses. Two of these lines are available for output (CTL0 and CTL1), and the other two are used as inputs (STI0 and STI1).

### Driving the Control Output Lines

Setting bits 0 and 1 of GPIO CONTROL register 2 places a logic low on CTL0 and CTL1, respectively. The definition of this CONTROL register is shown in the following diagram.

#### CONTROL Register 2 Peripheral Control

Most Significant Bit

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used					PSTS Error (1=Report; 0=Ignore)	Set CTL1 (1=Low; 0=High)	Set CTL0 (1=Low; 0=High)
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Ct10=0 ! Clear.

Ct11=1 ! Set.

CONTROL 12,2;Ct11\*2+Ct10

As indicated in the diagram, setting a bit in the register places the corresponding line Low, while clearing the bit places a logic High on the line. The logic polarity of these signals cannot be changed. The signal remains on these lines until another value is written into the CONTROL register, and Reset has no effect on the state of either line.



## Interrogating the Status Input Lines

The state of both status input lines STI0 and STI1 are determined by reading bits 0 and 1 of STATUS register 5, respectively. A logic “1” in a bit position indicates that the corresponding line is at logic Low, and a “0” indicates the opposite logic state. This logic polarity cannot be changed. The definition of GPIO STATUS register 5 is shown below.

**STATUS Register 5**      Peripheral Status

**Most Significant Bit**

**Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	PSTS Ok	EIR Line Low	STI1 Line Low	STI0 Line Low
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

```
STATUS 12,5;P_status  
Sti0=BIT(P_status,0)  
Sti1=BIT(P_status,1)
```

Reading this register returns a numeric value that reflects the logic states of these lines **at the instant the computer reads the interface lines**; the state of these lines are not latched by any internal or external event.

## Using the PSTS Line

The Peripheral Status line (PSTS) is generally used to indicate whether or not the peripheral device is functional. The current state of PSTS may be checked by reading STATUS Register 5 (bit 3). It may also optionally be checked automatically at the beginning of an OUTPUT or ENTER statement; normally, it is not checked. This feature is only enabled by by setting Bit 2 of CONTROL register 2.

### CONTROL Register 2    Peripheral Control

Most Significant Bit

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used					PSTS Error (1=Report; 0=Ignore)	Set CTL1 (1=Low; 0=High)	Set CTL0 (1=Low; 0=High)
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

When Bit 2 is set and PSTS is false at the beginning of either an OUTPUT or ENTER statement, Error 172 (**Peripheral error**) is reported. The error must be trapped with ON ERROR, since it generates no INTR or TIMEOUT branch.

---

## Summary of GPIO STATUS and CONTROL Registers

**STATUS Register 0**      Card Identification. Always 3.

**CONTROL Register 0**      Interface Reset. Any non-zero value causes a reset.

**STATUS Register 1**      Interrupt and DMA Status.

**Most Significant Bit**

**Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupts Are Enabled	An Interrupt Is Currently Requested	Interrupt Level Switches (Hardware Priority)	Interrupt Level Switches (Hardware Priority)	Burst-Mode DMA	Word-Mode DMA	DMA Channel 1 Enabled	DMA Channel 0 Enabled
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**CONTROL Register 1**      Set PCTL Line. Any non-zero value sets the line.

## STATUS Register 2

Most Significant Bit

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	Handshake In Process	Interrupts Are Enabled	Transfer In Progress
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

## CONTROL Register 2 Peripheral Control

Most Significant Bit

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used					PSTS Error (1=Report; 0=Ignore)	Set CTL1 (1=Low; 0=High)	Set CTL0 (1=Low; 0=High)
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1



**STATUS Register 3**      Data In (16 bits)

**CONTROL Register 3**      Data Out (16 bits)

**STATUS Register 4**      Interface Ready. Interface is Ready for a subsequent data transfer: 1=Ready, 0=Busy.

**STATUS Register 5**      Peripheral Status

**Most Significant Bit**

**Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	PSTS Ok	EIR Line Low	STI1 Line Low	STI0 Line Low
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**Interrupt Enable Register**      (ENABLE INTR)

**Most Significant Bit**

**Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used						Enable Interface Ready Interrupts	Enable EIR Interrupts
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

---

## Summary of GPIO READIO and WRITEIO Registers

This section describes the GPIO Interface's READIO and WRITEIO registers. Keep in mind that these registers should be used **only** when you know the exact consequences of their use, as using some of the registers improperly may result in improper interface behavior. If the desired operation can be performed with STATUS or CONTROL, you should not use READIO or WRITEIO.

### GPIO READIO Registers

Register 0	Interface Ready
Register 1	Card Identification
Register 2	Undefined
Register 3	Interrupt Status
Register 4	MSB of Data In
Register 5	LSB of Data In
Register 6	Undefined
Register 7	Peripheral Status

**READIO Register 0**      Interface Ready. A 1 indicates that the interface is Ready for subsequent data transfers, and 0 indicates Not Ready.

**READIO Register 1**      Card Identification. This register always contains 3, the identification for GPIO interfaces.

**READIO Register 3**      Interrupt Status

**Most Significant Bit**

**Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupts Are Enabled	An Interrupt Is Currently Requested	Interrupt Level Switches (Hardware Priority)	Interrupt Level Switches (Hardware Priority)	Burst-Mode DMA	Word-Mode DMA	DMA Channel 1 Enabled	DMA Channel 0 Enabled
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**READIO Register 4**      MSB of Data In**Most Significant Bit****Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI15	DI14	DI13	DI12	DI11	DI10	DI9	DI8
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**READIO Register 5**      LSB of Data In**Most Significant Bit****Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**READIO Register 7**      Peripheral Status**Most Significant Bit****Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	PSTS Ok	EIR Line Low	STI1 Line Low	STI0 Line Low
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

## GPIO WRITEIO Registers

WRITEIO Register 0	Set PCTL
WRITEIO Register 1	Reset Interface
WRITEIO Register 2	Interrupt Mask
WRITEIO Register 3	Interrupt and DMA Enable
WRITEIO Register 4	MSB of Data Out
WRITEIO Register 5	LSB of Data Out
WRITEIO Register 6	Undefined
WRITEIO Register 7	Set Control Output Lines

**WRITEIO Register 0** Set PCTL. Writing any non-zero numeric value to this register places PCTL in the Set state; writing zero causes no action.

**WRITEIO Register 1** Reset Interface. Writing any non-zero numeric value to this register resets the interface.

**WRITEIO Register 2** Interrupt Mask

**Most Significant Bit**

**Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used						Enable Interface Ready Interrupts	Enable EIR Interrupts
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1



**WRITEIO Register 3**      Interrupt and DMA Enable**Most Significant Bit****Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Enable Interrupts	Not used			Enable Burst-Mode DMA	Enable Word-Mode DMA	Enable DMA Channel 1	Enable DMA Channel 0
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**WRITEIO Register 4**      MSB of Data Out**Most Significant Bit****Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DO15	DO14	DO13	DO12	DO11	DO10	DO9	DO8
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**WRITEIO Register 5**      LSB of Data Out**Most Significant Bit****Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**WRITEIO Register 7**      Set Control Output Lines**Most Significant Bit****Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used						Set CTL1 (1=Low; 0=High)	Set CTL0 (1=Low; 0=High)
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

# Table of Contents

---

## Chapter 16: HP-HIL Interface

The Interface to HP-HIL Devices .....	16-1
Preview of HP-HIL Devices .....	16-3
Communicating through the HP-HIL Interface .....	16-4
Supported HP-HIL Devices.....	16-6
Selecting HP-HIL Devices .....	16-6
Identifying All Devices on the HP-HIL Link .....	16-8
Explanation of the HIL_ID Program.....	16-10
HP-HIL Devices .....	16-22
Communicating with HP-HIL Devices .....	16-31
HP-HIL Device Characteristics .....	16-31
ID Module .....	16-32
Function Box .....	16-35
Using a Touchscreen.....	16-46
Using a Bar Code Reader .....	16-51
Interaction Among Multiple HP-HIL Devices .....	16-55



## HP-HIL Interface

### The Interface to HP-HIL Devices

This chapter describes communication with the HP-HIL interface. It is primarily a description of the use of HIL SEND and HIL EXT to handle the HP-HIL interface. This interface is capable of supporting up to seven devices, such as a Function Box, a system ID Module, and other peripherals generally related to human input.

Before launching into a discussion of the workings of the HP-HIL interface, a general overview needs to be presented. HP-HIL stands for "Hewlett-Packard Human Interface Link". The following diagram illustrates the basic components.

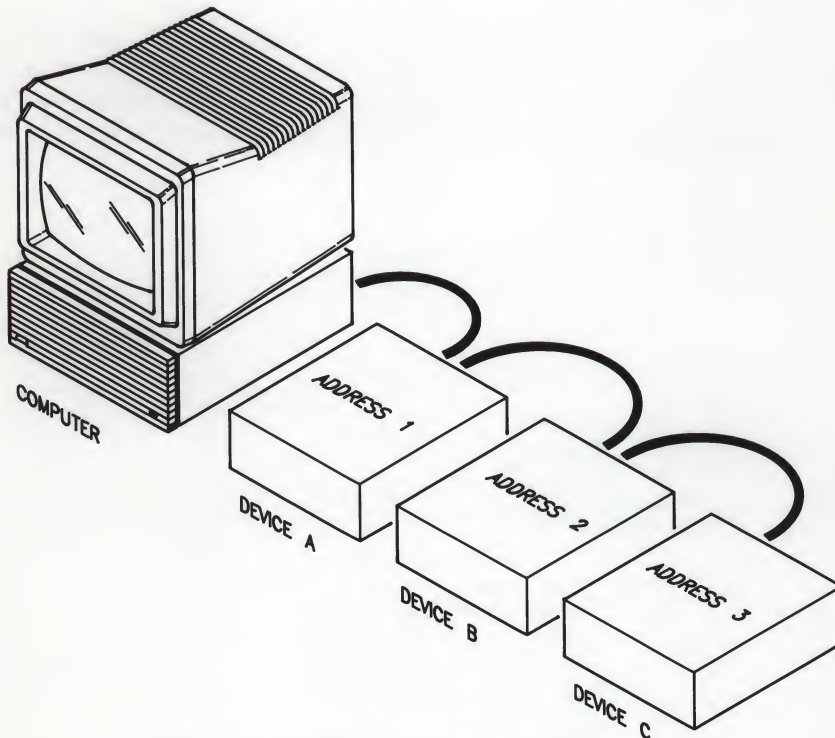


Figure 16-1. Hewlett-Packard Human Interface Link



HP-HIL initialization takes place when BASIC is booted or when you execute SCRATCH A. BASIC logs HP-HIL devices which are present on the link. The link can deal with a maximum of seven devices at any one time (any devices present after the seventh one are not recognized). If you add an HP-HIL device to the HP-HIL link after the BASIC system has been booted, the device will not automatically be recognized by the system. If you replace a device on the link with a different device, the system may mis-interpret the data coming from the new device. Therefore, when adding, removing, or replacing a device on the link, either re-boot the system or execute a SCRATCH A.

The address of a particular device is merely its topological order of placement along the link. In the above diagram, Device A has address 1, B has address 2, and C has address 3. This is only a result of their physical order of connection. If Device C had been connected between Devices A and B, Device A would still have been address 1, but Device C would be address 2, and B would be address 3. The type of device is irrelevant to the address assigned to it.

After the link is operational, and during subsequent link operations, each device looks at the data being sent down the link. If a device notices that the destination address associated with the link data is the same as that device's address, that device receives and acts on the data. Otherwise, the data is merely shuttled along to the next device.

## Preview of HP-HIL Devices

HP-HIL devices can be divided into a number of categories. This section provides you with a table that includes these categories, as well as a list of high level and low level statements that apply to each category.

**Table 16-1. HP-HIL Device categories**

<b>HP-HIL Device Categories</b>	<b>High Level BASIC Access</b>	<b>Low Level BASIC Access</b>
HP-HIL Keyboards	BASIC Operating System normally handles keystrokes. Programs can enter text and numbers with the statements: INPUT, LINPUT, and ENTER.	ON/OFF KEY ON/OFF KBD KBD\$
Relative Positioner	BASIC Operating System handles as cursor-movement input. Can also be used with GRAPHICS INPUT IS.	ON/OFF KBD (traps movement as arrow keys and also traps mouse buttons) KBD\$ ON/OFF KNOB ON/OFF CDIAL CDIAL
Absolute Positioner	Can be used with GRAPHICS INPUT IS.	HIL SEND ON HIL EXT HILBUF\$
ID Module	One can be used with SYSTEM\$("SERIAL NUMBER")	HIL SEND ON HIL EXT HILBUF\$
Other Devices	None	HIL SEND ON HIL EXT HILBUF\$

## Communicating through the HP-HIL Interface

This section provides a brief description of the HP-HIL Interface Driver. The HP-HIL Interface Driver supports a set of statements which allow communications between the HP-HIL interface and HP-HIL devices. These statements and commands are as follows:

`HIL SEND Address; HIL_Command`

allows you to send HP-HIL *Commands* to an HP-HIL device (e.g. `HIL SEND 1;IDD`). The BASIC HP-HIL commands can be found in the "HP-HIL Appendix" in this manual. The *Address* is the location of the device in the HP-HIL link. Address 1 is assigned to the first device on the link that is addressable (i.e. any device except HP-HIL Extension Modules). Ascending address are assigned to subsequent devices on the link.

`ON HIL EXT Address_mask Branch`

enables end-of-line interrupts from HP-HIL devices. This statement allows you receive interrupts from up to seven devices on the HP-HIL link. The *Address\_mask* is a bit-map of the locations of the device or devices in the HP-HIL link. The default *Address\_mask* is 254 which allows up to 7 devices to send interrupts. To select devices from which you want to receive interrupts, you need to raise 2 to the power of that device's address. For example, if the device is the second one on the HP-HIL link then you would raise 2 to the 2nd power which would result in an *Address\_mask* of 4. If you have two devices on the HP-HIL link, one at the second position and the other at the third position, then to enable interrupts from both of these devices you would add  $2^2$  and  $2^3$  together. The resulting *Address\_mask* would be 12. *Branch* refers to a branch to a program line number, label, subroutine or subprogram using the keywords `GOTO`, `GOSUB`, `RECOVER`, or `CALL`.

`OFF HIL EXT`

disables end-of-line interrupts. This statement **does not** require an address mask. It will disable all previously enabled end-of-line interrupts for HP-HIL devices.



## HILBUF\$

is a function used to capture data returned from HP-HIL devices. This function provides a buffer for data to be stored in after execution of the first two statements listed above. Note that this buffer only holds up to 256 bytes of data. Once the buffer limit is reached it will not receive any new data until it has been emptied by reading the buffer. The first byte stored in the string buffer tells you if data has been lost. This byte is initially zero (the "null" character). It is only zero if no data has been lost; otherwise, it contains the number of packets lost. A packet is three or more bytes consisting of the packet length (first byte), the device address (second byte), and one or more bytes of data from the device (i.e. a poll record or a response to a command). A *poll record* is a set of data sent by an HP-HIL device to HILBUF\$ which accompanies an ON HIL EXT interrupt. This data first includes a poll record header byte which contains information about the bytes that follow it (covered in the "HP-HIL Appendix" under the section entitled "Poll Record").

The poll records (see HILBUF\$ above) for the devices listed below are not available through ON HIL EXT. An *Address\_mask* (ON HIL EXT) can include these devices, but no interrupts will be generated. The excluded devices are:

- any relative pointing device
- the current GRAPHICS INPUT IS device
- any system keyboard

The HIL SEND statement operates under a different set of conditions than ON HIL EXT:

- HIL SEND *Address*;IDD is allowed with all devices.
- HIL SEND *Address*;HP-HIL *command* is allowed if that device is not:
  - a relative pointing device
  - currently a GRAPHICS INPUT IS device



---

## Supported HP-HIL Devices

This section provides a brief description of those devices supported by the HIL Interface Driver, references to information for those devices not supported by the HIL Interface Driver, a program for identifying all devices on the HP-HIL link, and an explanation of that program. The topics are as follows:

- Selecting HP-HIL Devices
- Identifying All Devices on the HP-HIL Link
- Explanation of the HIL\_ID Program
- HP-HIL Devices

### Selecting HP-HIL Devices

When you enter BASIC/UX with the *rmb* command, the BASIC/UX system uses all available devices on the HP-HIL link. This prevents other HP-UX processes from accessing these HP-HIL devices. However, HP-HIL devices can be selected by using the **SET HIL MASK** command. This allows you to select only those HP-HIL devices required for your particular BASIC/UX task and relinquishes the remaining devices for use by other HP-UX processes. This section explains the use of this command.

### Enabling HP-HIL Devices

The **SET HIL MASK** statement enables the specified HP-HIL device for use by the BASIC/UX system. The syntax for this command is as follows:

```
SET HIL MASK address_mask
```

where the *address\_mask* is obtained by raising 2 to the power of each of the addresses of the desired devices and adding these values.

### Example

The following program uses the HP-HIL identify and describe command to determine the type of device that is located at address 3 in the link and gives the device's characteristics. The HP-HIL device shown below has address 3 assigned to it because it is the third device in the HP-HIL link.

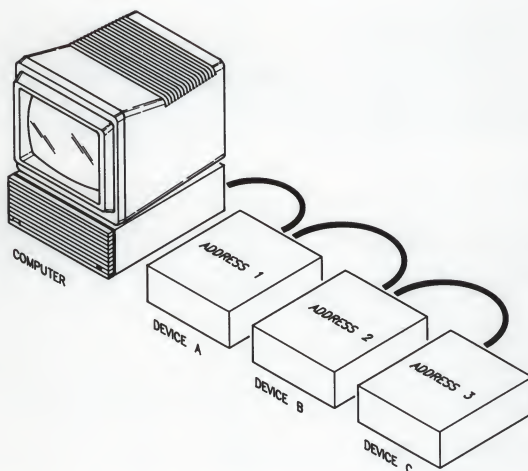


Figure 16-2. An HP-HIL Device with an Address of 3

The program then assigns the device to the current process. This prevents another process from using the device. Finally it prints the device's identification number and characteristics on the screen. You can use the "Device ID Byte Definitions" table in the "HP-HIL Appendix" in this manual and the identification number to determine the name of your device.

```
100 SET HIL MASK 2^3           ! Assigns the device at address 3 to the
110                               ! current BASIC/UX process (locks it).
120 HIL SEND 3;IDD ! Sends the IDD command to the device at address 3.
130 A$=HILBUF$           ! Assigns the IDD information found in the HP-HIL
140                       ! buffer string function to the string A$.
150 FOR I=1 TO LEN(A$)     ! This FOR loop prints the contents
160   B$=IVAL$(NUM(A$(I)),16) ! of A$ as hexadecimal values. The
170   PRINT B$(3);" ";      ! fourth element is the ID number
180 NEXT I                 ! that identifies the device.
190 END
```

## Identifying All Devices on the HP-HIL Link

Each device in the HP-HIL link has a Device ID which identifies that device and a Describe Record which provides you with device characteristics. This information can be obtained by executing the HP-HIL `IDD` command and parsing the string value returned by the `HILBUF$` function. A program called `HIL_ID`, located in the directory called `/usr/lib/rmb/demo`, makes use of the `IDD` command and `HILBUF$` function for:

- Determining if a device is recognized as being in the HP-HIL link,
- Identifying the device at a specific address, and
- Determining the device's characteristics (what it can do).

Assuming your HP-HIL link has a:

- **Touchscreen** located at address 1
- **ITF Keyboard** (HP 46020/21A) located at address 2
- **Function Box** located at address 3

Executing the `HIL_ID` program would produce the following output:

```
HP 35723A (Touchscreen) located at address 1
Describe Record Information
  I/O Descriptor Information
    Does not support Prompts/Acknowledges 1 thru 7
    Supports Proximity Detection
    Does not report buttons
  X and Y axis information reported
    Absolute positioning device
    Returns 8 bits/axis
```

```
HP 46020/21A (ITF Keyboard) located at address 2
Describe Record Information
  No special features
```

```
HP 46086A (Function Box) located at address 3
Describe Record Information
  I/O Descriptor Information
    Recognizes General Prompt and Acknowledge
    Does not support Prompts/Acknowledges 1 thru 7
    Does not report buttons
  No axis information reported
```

NO MORE DEVICES.

Just what does the above information tell you? Let's look at the first device. It is a **Touchscreen** located at address 1 in the HP-HIL link. To determine what this device can do, you need to know its characteristics. The Describe Record provides you with this information. Describe Record information returned by this device is as follows:

- I/O Descriptor byte information is reported. The information supplied in this byte tells you that when you touch your finger on the screen it will be detected and when you remove it from the screen it will be detected. This is called proximity in/out detection.
- It is an absolute positioning device. This means that every coordinate position on the screen is referenced to the lower left-hand corner of the Touchscreen (X-coordinate = 0 and Y-coordinate = 0).
- X and Y axis information is reported. This tells you that Poll Records received when communicating with this device will contain X and Y coordinate information. These are absolute coordinate positions.
- Coordinate information is returned as 8 bits per axis. This means there will only be one byte of X coordinate information returned in the Poll Record and one byte of Y coordinate information returned in the Poll Record.

Putting the above information together, the Touchscreen makes a great device for option selection from screen menus. Other uses are left up to your creativity.



## Explanation of the HIL\_ID Program

The program called HIL\_ID is located in the directory called /usr/lib/rmb/demo. This program has been divided into four segments. Each segment of the program will be given and explained in this section. It is not absolutely necessary for you to read this section to gain an understanding of how to communicate with HP-HIL devices. If you decide not to read this section, skip to the next one entitled "HP-HIL Devices."

### Segment 1 of HIL\_ID

This segment of the program executes the HP-HIL command IDD for each device in the HP-HIL link. The system places the information returned from executing this command in the string buffer used by the HILBUF\$ function. The buffer information has been stored in a string array to simplify future processing. The string buffer used by HILBUF\$ is cleared each time the function is executed.

```
1000 OPTION BASE 1
1010 DIM Idd$(7) [20]
1020 INTEGER Dev_id,Address_num,Des_header,Test,Count
1030 COM INTEGER Io_header
1040 !
1050 !*****
1060 ! This is segment 1 of the program. It stores Identify
1070 ! and Describe information in the array Idd$.
1080 !*****
1090 !
1100 ON ERROR GOTO Link_end
1110 FOR I=1 TO 7
1120     HIL SEND I;IDD
1130     Idd$(I)=HILBUF$
1140 NEXT I
1150 Link_end: !
1160 Count=I-1
1170 PRINT
1180 !
```

The following information is an explanation of program segment 1.

Line **1000** sets a lowerbound of 1 for the string array **Ida\$** in the program.

Lines **1010** to **1030** declare the variables for the program.

Line **1100** sets up a branch to the label **Link\_end** if an error occurs. Note that an error will occur in the FOR loop of lines **1110** to **1140** if there are less than 7 devices on the HP-HIL link. The branch to the label **Link\_end** is designed to prevent execution of the program from stopping when this condition occurs.

Lines **1110** to **1140** are a FOR loop which executes the HP-HIL **IDD** command as many times as there are devices recognized plus one or until seven devices have been recognized on the HP-HIL link. Identify and Describe information from executing the **IDD** command is stored in the string array **Ida\$** for future processing. If there are fewer than 7 HP-HIL devices in the link, then **ON ERROR** causes a branch to be taken outside of the FOR loop. If there are 7 devices in the loop, then **ON ERROR** branching **does not** take place.

Line **1150** is the destination label **Link\_end** for the **ON ERROR** branch from the FOR loop.

Line **1160** sets the loop **Count** for the FOR loop in the second segment of the demonstration program.

## Segment 2 of HIL\_ID

This segment of the program is a large FOR loop with a SELECT statement in it for selecting and identifying the various devices in the HP-HIL link. Each CASE statement within the SELECT statement causes a message to be displayed for the device found on the link. This message contains the HP product number, device name and device address. After the SELECT statement a CALL is made to the subprogram Describe\_rec which is used for determining the characteristics of the devices in the HP-HIL link. If the device is found to report I/O Descriptor information, the Describe\_rec subprogram calls the subprogram Io\_descriptor. This subprogram provides additional information about the device.

```
1190 !*****
1200 ! This is segment 2 of the program. It identifies all
1210 ! devices on the link and provides their address
1220 ! in the link. It also uses two subprograms called
1230 ! Describe_rec and Io_descriptor. These subprograms
1240 ! describe what each device can do.
1250 !*****
1260 !
1270 FOR I=1 TO Count
1280   Dev_id=NUM(Idd$(I)[4])
1290   PRINT
1300   !
1310   Address_num=NUM(Idd$(I)[3])
1320   Des_header=NUM(Idd$(I)[5])
1330   Io_header=NUM(Idd$(I)[LEN(Idd$(I))])
1340   !
1350   SELECT Dev_id
1360   CASE 48 ! Function Box
1370     PRINT "HP 46086A (Function Box) located at address ";Address_num
1380   CASE 52 ! ID Module
1390     PRINT "HP 46084A (HP-HIL ID Module) located at address ";Address_num
1400   CASE 92 ! Bar Code Reader
1410     PRINT "HP 92916A (Bar Code Reader) located at address ";Address_num
1420   CASE 96 ! Rotary Control Knob
1430     PRINT "HP 46083A (Rotary Control Knob) located at address ";Address_num
```

```

1440 CASE 97 ! Control Dials and Quadrature Port
1450     ! This is a test to determine if the device is an HP 46085A
1460     ! or an HP 46094A. Note that both of these devices have the
1470     ! same ID number.
1480     IF (NOT BIT(Io_header,0)) THEN
1490         PRINT "One third of an HP 46085A (Control Dials) located at address
            ";Address_num
1500     ELSE
1510         PRINT "HP 46094A (HP-HIL Quadrature Port) located at address
            ";Address_num
1520     END IF
1530 CASE 104 ! HP Mouse
1540     PRINT "HP 46060A (HP Mouse) located at address ";Address_num
1550 CASE 140 ! Touchscreen
1560     PRINT "HP 35723A (Touchscreen) located at address ";Address_num
1570 CASE 147 ! Digitizer A
1580     PRINT "HP 46087A (Digitizer A) located at address ";Address_num
1590 CASE 148 ! Digitizer B
1600     PRINT "HP 46088A (Digitizer B) located at address ";Address_num
1610 CASE 149 ! Graphics Tablet
1620     PRINT "HP 45911A (Graphics Tablet) located at address ";Address_num
1630 CASE 160 TO 191 ! Integral Keyboard
1640     PRINT "Integral Keyboard located at address ";Address_num
1650 CASE 192 TO 223 ! HP 46020/21A Keyboard
1660     PRINT "HP 46020/21A (ITF Keyboard) located at address ";Address_num
1670 CASE ELSE
1680     PRINT "Un-recognized device located at address ";Address_num
1690     Unknown_dev$=IVAL$(Dev_id,16)
1700     PRINT "Device ID is ";Unknown_dev$[3]
1710 END SELECT
1720 CALL Describe_rec(Des_header,Address_num,Dev_id)
1730 !
1740 NEXT I
1750 PRINT
1760 PRINT "NO MORE DEVICES."
1770 !
1780 END
1790 !

```



The following information is an explanation of program segment 2.

Lines **1270** to **1740** are a FOR loop which identifies all the devices specified by the **Count** variable. It also gives the address of these devices in the HP-HIL link.

Lines **1350** to **1710** are a **SELECT** statement within the FOR loop. This statement contains **CASE** statements which cause a message to be printed for each type of device found on the HP-HIL link. An example of the type of message printed is as follows:

```
HP 46086A (Function Box) located at address 2
```

Assuming that there is a Function Box located at address 2 in your HP-HIL link.

Note that in lines **1440** to **1520** a test is made for the type of device found with the ID number of 97 (decimal) because there are two devices which have that device ID number. They are the Control Dials box and the Quadrature Port. Also, in the case of a device not being recognized, lines **1670** to **1700** will cause the following message to be displayed:

```
Un-recognized device located at address 2  
Device ID is 20
```

Assuming there is an un-recognized device located at address 2 and the ID number of that device is 20 (hexadecimal).

### Segment 3 of HIL\_ID

This segment of the program is a subprogram called `Describe_rec`. The subprogram interprets each bit of the Describe Record byte to characterize the device at a specified address in the HP-HIL link. For a detail description of the Describe Record byte, read the section in the "HP-HIL Appendix" entitled "Describe Record".

```
1800 !*****
1810 ! This is segment 3 of the program. It is a subprogarm
1820 ! that provides information on each device. This
1830 ! information will help you determine what you can do
1840 ! with a particular device.
1850 !*****
1860 !
1870 SUB Describe_rec(INTEGER Des_header,Address_num,Dev_id)
1880   COM INTEGER Io_header
1890 !
1900   PRINT TAB(2),"Describe Record Information"
1910   IF Des_header=0 THEN
1920     PRINT TAB(5),"No special features"
1930     SUBEXIT
1940   END IF
1950   IF BIT(Des_header,2) THEN PRINT TAB(5),"Reports Security Code
information"
1960   IF BIT(Des_header,3) THEN PRINT TAB(5),"Supports the Extended Describe
command"
1970   !
1980   IF BIT(Des_header,4) THEN CALL Io_descriptor(Io_header)
1990   IF BIT(Des_header,7) THEN PRINT TAB(5),"Contains two independent sets of
coordinate axes"
2000   !
2010   SELECT Des_header MOD 4
2020   CASE 0
2030     PRINT TAB(5),"No axis information reported"
2040     SUBEXIT
2050   CASE 1
2060     PRINT TAB(5),"X axis information reported"
2070   CASE 2
2080     PRINT TAB(5),"X and Y axis information reported"
2090   CASE 3
2100     PRINT TAB(5),"X, Y and Z axis information reported"
2110   END SELECT
2120   !
```

```

2130   IF BIT(Des_header,6) THEN
2140       PRINT TAB(8),"Absolute positioning device"
2150   ELSE
2160       PRINT TAB(8),"Relative positioning device"
2170   END IF
2180   IF BIT(Des_header,5) THEN
2190       PRINT TAB(8),"Returns 16 bits/axis"
2200   ELSE
2210       PRINT TAB(8),"Returns 8 bits/axis"
2220   END IF
2230 SUBEND
2240 !

```

The following information is an explanation of program segment 3.

Line **1900** prints the message:

```
Describe Record Information
```

This indicates that the information to follow this messages was taken from the Describe Record. Note that the Describe Record consist of up to 10 bytes of information. This information includes a Describe Record Header, Axes information, and an I/O Descriptor Byte.

Lines **1910** through **1940** are an IF...THEN statement which tests to see if there is information in the Describe Record Header byte . If the byte contains all zeros, then the following message is printed and the subprogram is exited:

```
No special features
```

If the Describe Record Header byte **is not** all zeros, then the subprogram continues to process the header information.

Line **1950** tests bit 2 of the Describe Record Header byte to determine if the HP-HIL device reports security code information. If it **does not**, then the subprogram passes on to the next test. However, if it does report security code information, then the following message is displayed:

**Reports Security Code information**

Line **1960** tests bit 3 of the Describe Record Header byte to determine if the HP-HIL device supports the Extended Describe the command. If it **does not**, then the subprogram passes on to the next test. However, if it does support the Extended Describe command, then the following message is displayed:

**Supports the Extended Describe command**

Line **1980** tests bit 4 of the Describe Record Header byte to determine if the HP-HIL device reports I/O Descriptor information. If it **does not**, the subprogram passes on to the next test. However, if it does a **CALL** to the subprogram called **Io\_descriptor** is made and I/O Descriptor byte information is displayed.

Line **1990** tests bit 7 of the Describe Record Header byte to determine if the HP-HIL device contains two independent sets of coordinate axes. If it **does not**, the subprogram passes on to the next test. However, if it does the following information is displayed and the subprogram passes on to the next test:

**Contains two independent sets of coordinate axes**

Lines **2010** through **2110** are a **SELECT** statement which test bits 0 and 1 for axis information. If no axis information is reported, then the following message is displayed and the subprogram is exited.

**No axis information reported**

However, if axis information is reported, you may receive one of the following messages:

**X axis information reported**

**X and Y axis information reported**

**X, Y, and Z axis information reported**

Once this test is completed program flow passes on to the next statement.



Lines **2130** through **2170** test bit 6 of the Describe Record Header byte to determine if the device is a relative or absolute positioning device. If bit 6 is set, the following message is displayed:

**Absolute positioning device**

If bit 6 is clear (not set), the following message is displayed:

**Relative positioning device**

Lines **2180** through **2220** complete the subprogram called Describe\_rec. They test bit 5 of the Describe Record Header byte to determine if the device returns 8 bits per axis or 16 bits per axis. If bit 5 is set, the following message is displayed:

**Returns 16 bits/axis**

If bit 5 is clear (not set), the following message is displayed:

**Returns 8 bits/axis**

## Segment 4 of HIL\_ID

This segment of the program is a subprogram called `Io_descriptor`. This subprogram provides information on the number of buttons the devices has and whether or not the device responds to general prompt and acknowledge commands. It also provides information on whether or not the device supports "proximity detection". Proximity detection checks for the in and out motion of a stylus or finger in relation to a digitizer or touchscreen. `Io_descriptor` is called by the subprogram `Describe_rec`.

```
2250 !*****
2260 ! This is segment 4 of the program. It is a
2270 ! subprogram that provides you with additional
2280 ! information on what a device can do.
2290 !*****
2300 !
2310 SUB Io_descriptor(INTEGER Io_header)
2320     PRINT TAB(5),"I/O Descriptor Information"
2330     IF Io_header=0 THEN
2340         PRINT TAB(8),"No features"
2350         SUBEXIT
2360     END IF
2370     IF BIT(Io_header,7) THEN PRINT TAB(8),"Recognizes General Prompt and
Acknowledge"
2380     !
2390     Test_bits=(Io_header MOD 128) DIV 16
2400     SELECT Test_bits
2410     CASE 0
2420         PRINT TAB(8),"Does not support Prompts/Acknowledges 1 thru 7"
2430     CASE 1
2440         PRINT TAB(8),"Supports Prompt/Acknowledge 1"
2450     CASE 2
2460         PRINT TAB(8),"Supports Prompts/Acknowledges 1 and 2"
2470     CASE ELSE
2480         PRINT TAB(8),"Supports Prompts/Acknowledges 1 thru "&VAL$(Test_bits)
2490     END SELECT
2500     !
2510     IF BIT(Io_header,3) THEN PRINT TAB(8),"Supports Proximity Detection"
2520     !
2530     Test_bits=(Io_header MOD 8)
2540     SELECT Test_bits
2550     CASE 0
2560         PRINT TAB(8),"Does not report buttons"
2570     CASE 1
2580         PRINT TAB(8),"Reports 1 button"
2590     CASE 2
2600         PRINT TAB(8),"Reports buttons 1 and 2"
2610     CASE ELSE
2620         PRINT TAB(8),"Reports buttons 1 thru "&VAL$(Test_bits)
2630     END SELECT
2640 SUBEND
```

The following information is an explanation of program segment 4.

Line **2320** displays the following message:

**I/O Descriptor Information**

Lines **2330** through **2350** test the I/O Descriptor byte to determine if it contains any information. If this byte **is not** all zeros, then the subprogram continues on with the next test. However, if it **does** contain all zeros, then the message given below is displayed and program execution is passed to the subprogram (**Describe\_rec**) which called it.

**No features**

Line **2370** tests bit 7 of the I/O Descriptor byte (**Io\_header**) to determine if the device being tested recognizes General Prompt and Acknowledge commands. A message is displayed if the test is true; otherwise, no message is displayed and the subprogram continues with the next test. The message displayed is:

**Recognizes General Prompt and Acknowledge**

Line **2390** does a MOD 128 of the I/O Descriptor byte to mask off the bits above bit 6 of the byte and then does a DIV 16 to mask off the lower four bits of the byte and shifts the remaining bits to the right. The variable **Test\_bits** is assigned the result of the above operation. The result is a value in the range of 0 through 7. Note that **Test\_bits** is the decimal value of bits 4 through 6 of the IO Descriptor byte.

Lines **2400** through **2490** are a **SELECT** statement which tests bits 4 through 6 of the I/O Descriptor byte to determine if Prompts/Acknowledges 1 through 7 are supported by the device being tested. If they **are not** supported, the following message is displayed:

**Does not support Prompts/Acknowledges 1 thru 7**

If they are supported, you will receive one of the following messages depending upon how many Prompts/Acknowledges your device supports:

Supports Prompt/Acknowledge 1

Supports Prompts/Acknowledges 1 and 2

Supports Prompts/Acknowledges 1 thru 3

Supports Prompts/Acknowledges 1 thru 4

Supports Prompts/Acknowledges 1 thru 5

Supports Prompts/Acknowledges 1 thru 6

Supports Prompts/Acknowledges 1 thru 7

Line **2510** tests bit 3 of the I/O Descriptor byte to determine if the device being tested supports proximity detection. A message is displayed if the test is true; otherwise, no message is displayed and the subprogram continues with the next test. The message displayed is:

Supports Proximity Detection

Line **2530** does a MOD 8 of the I/O Descriptor byte to mask off the bits above bit 2 the byte. The variable **Test\_bits** is assigned the result of the above operation. The result is a value in the range of 0 through 7. Note that **Test\_bits** is the decimal value of bits 0 through 2 of the IO Descriptor byte.

Lines **2540** through **2630** are a **SELECT** statement which tests bits 0 through 2 of the I/O Descriptor byte to determine if Buttons 1 through 7 are reported by the device being tested. If they **are not** reported, the following message is displayed:

Does not report buttons



If they are reported, you will receive one of the following messages depending upon how many buttons your device reports:

Reports 1 button

Reports buttons 1 and 2

Reports buttons 1 thru 3

Reports buttons 1 thru 4

Reports buttons 1 thru 5

Reports buttons 1 thru 6

Reports buttons 1 thru 7

## HP-HIL Devices

A brief description will be provided for those devices supported by HIL SEND, ON HIL EXT, and HILBUF\$. For those devices not supported by these statements and function, there will be a reference given to help you locate further information on that device, as well as statements you may use to interact with it.

HP-HIL devices have been divided into the following categories:

- HP-HIL Keyboards
- Relative Positioners
- Absolute Positioners
- Security Device (the ID Module)
- Other Devices (i.e. Keyboards, Button Devices, Bar Code Reader)

## HP-HIL Keyboards

There are three HP-HIL keyboards supported (as Keyboards) on the HP-HIL link. They are the:

- HP 46020/21A (for information on this keyboard see the manual entitled *Using the BASIC/UX System*)
- HP 98203C (this keyboard is not supported on BASIC/UX)
- Integral Keyboard (for information on this keyboard see the *HP-UX Technical BASIC Getting Started Guide*)

These keyboards will not cause ON HIL EXT interrupts. To do interrupt branching with the keyboard keys, you need to use the following statements and function:

ON/OFF KEY	ON KEY defines and enables an event-initiated branch to be taken when a softkey is pressed. OFF KEY cancels event-initiated branches previously defined and enabled by an ON KEY statement. Without the KBD binary, subsequent softkey presses cause beeps. With the KBD binary, the action of subsequent softkey presses depends upon the typing-aid definitions.
ON/OFF KBD	ON KBD defines and enables an event-initiated branch to be taken when a key is pressed. OFF KBD cancels event-initiated branches previously defined and enabled by an ON KBD statement. Subsequent key presses are sent to the operating system in the normal manner.
KBD\$	This function returns the contents of the keyboard buffer when ON KBD is active.

For more information on keyboards, read the chapter "Keyboard Reference" found in the *Using the BASIC/UX System* manual, as well as the chapter "Keyboard Interfaces" found in the *BASIC/UX Interfacing Techniques* manual.

## Relative Positioners

These devices will not cause ON HIL EXT interrupts. Using HIL SEND to transmit a command (other than IDD) to one of these devices will cause an error. Relative positioners can be categorized into two groups: those that are two axis devices and those that are three axis devices. A list of these devices and their statements and functions is given below.

Examples of two-axis relative positioning devices are:

- **HP 46060A/B (HP-Mouse)**
- **HP 46083A (Rotary Control Knob)**
- **HP 46094A (HP-HIL/Quadrature Port)**
- **HP 98203C (this keyboard is not supported on BASIC/UX)**

These devices support the following statements and functions. Note that in the case of the HP 46094A (HP-HIL/Quadrature Port) it supports statements and functions appropriate to the quadrature device connected to it (e.g. the HP 46095A 3-button Mouse).

<b>ON/OFF KBD</b>	<b>ON KBD</b> defines and enables an event-initiated branch to be taken when a key is pressed. <b>OFF KBD</b> cancels event-initiated branches previously defined and enabled by an <b>ON KBD</b> statement. Subsequent key presses are sent to the operating system in the normal manner.
-------------------	--

KBD\$	This function returns the contents of the keyboard buffer.
ON/OFF KNOB	ON KNOB defines and enables an event-initiated branch to be taken when the knob is turned. OFF KNOB cancels event-initiated branches previously defined and enabled by an ON KNOB statement. Subsequent use of the knob results in normal scrolling or cursor movement.
KNOBX and KNOBY	return the counts accumulated for X and Y motions of the relative positioning devices.
DIGITIZE	This statement is used when:  GRAPHICS INPUT IS KBD,"KBD"  It inputs the X and Y coordinates of a digitized point from the locator specified by GRAPHICS INPUT IS.
READ LOCATOR	This statement is used when:  GRAPHICS INPUT IS KBD,"KBD"  It samples the locator device, without waiting for a digitizing operation.

For more information on these statements, read the *BASIC Language Reference*, the chapter entitled "Keyboard Interfaces" found in the *BASIC/UX Interfacing Techniques* manual, and the chapter entitled "Interactive Graphics and Graphics Input" found in the *BASIC Graphics Techniques* manual.



There are also three-axis devices, for example the **HP 46085A (Control Dials)** module contains 3 such devices. This device supports the following statements and function:

**ON/OFF CDIAL**

**ON CDIAL** enables end-of-line interrupts in response to the rotation of one or more knobs on the HP-HIL Control Dials device. While such interrupts are enabled, pulses (rotation counts) are accumulated and returned via the **CDIAL** function (see below). **OFF CDIAL** cancels end-of-line interrupts previously enabled by an **ON CDIAL** statement. After an **OFF CDIAL** statement, left over counts may be read via **CDIAL** (but only once), and no further accumulation occurs.

**CDIAL (*Counter*)**

This function is used to return counts from the Control Dials module or other 3-axis relative positioning devices. It is linked to a status word and 15 *Counters*. Each of the 15 high order bits in the status word corresponds to one *Counter*, the first *Counter* being represented by bit 1 of the status word (bit 0 of the status word is unused). Normally the *Counters* one through nine correspond to the nine knobs on the Control Dials module. *Counter 1* is the knob in the lower left-hand corner of the module. The remaining *Counters* are numbered from left to right. The status word and *Counters* are zeroed when an **ON CDIAL** statement is executed. Thereafter, whenever a count arrives from any of the knobs, the corresponding *Counter* is incremented and its status bit is set. Reading a *Counter* zeros both the *Counter* and its bit in the status word. Reading the status word does not change its value. The status word is read as **CDIAL(0)**.

For more information on these statements see the *BASIC Programming Techniques* manual.

Note that when ON CDIAL interrupts are disabled, three-axis devices may be used with the two-axis statements and functions.

### **Absolute Positioners**

These devices can generate ON HIL EXT interrupts, but will not when:

```
GRAPHICS INPUT IS KBD,"TABLET"
```

is in effect. Moreover, due to the speed which data is returned from the digitizers, a BASIC program cannot keep up with them when using ON HIL EXT (HILBUF\$ overflows). Therefore, the only device in this group capable of using the ON HIL EXT statement is the Touchscreen. Using HIL SEND to transmit a command (other than IDD) to these devices while:

```
GRAPHICS INPUT IS KBD,"TABLET"
```

will result in an error.

The following statements should be used when:

```
GRAPHICS INPUT IS KBD,"TABLET"
```

is in effect.

<code>DIGITIZE X_coord,Y_coord</code>	inputs the <i>X</i> and <i>Y</i> coordinates of a digitized point.
---------------------------------------	--

<code>READ LOCATOR X_coord,Y_coord</code>	samples the locator device without waiting for a digitize operation.
---	--

For more information on these statements read the chapter "Interactive Graphics and Graphics Input" found in the *BASIC Graphics Techniques* manual. An explanation of each of these statements may also be found in the *BASIC Language Reference*.

The following are absolute position devices:

- **HP 35723A (HP-HIL/Touchscreen)** — This module is a screen bezel which replaces the bezel of the HP 35731 (medium resolution black and white) and HP 35741 (medium resolution color) 12-inch video monitors. It can be programmed to select various functions by simply touching the screen. Note that this device is simply a lower resolution digitizer. The Touchscreen can be used as a **GRAPHICS INPUT IS** device or with the **ON HIL EXT** statement.
- **HP 45911A (11 × 11 Graphics Tablet)** — This device is best used as a **GRAPHICS INPUT IS** device.
- **HP 46087A (A-size Digitizer)** — This device is best used as a **GRAPHICS INPUT IS** device.
- **HP 46088A (B-size Digitizer)** — This device is best used as a **GRAPHICS INPUT IS** device.

If a three-axis absolute positioning device existed, it could always be used with **HIL SEND** and **ON HIL EXT** since it would not be recognized for use with:

**GRAPHICS INPUT IS KBD,"TABLET"**

### **Security Device**

The **HP 46084A (HP-HIL ID Module)** is an HP-HIL device that returns an identification number for identifying you as the computer user. The identification number is unique to your particular ID Module. This allows application programs to use the ID Module to control access to program functions, data bases, and networks. Note that the identification number is the product/exchange and serial numbers returned in a packed format as explained in the section "ID Module" found in this chapter.

This device can be used with **SYSTEM\$(“SERIAL NUMBER”)** or **HIL SEND *device address*;RSC**.

## Other Devices

These devices can generate ON HIL EXT interrupts and respond to various HIL SEND commands. They all have HP-HIL device IDs less than 96 (60 hexadecimal).

The **HP 46086A (Function Box)** provides 32 keys to select software-defined functions. It has an LED that acts as a visual prompt for any purpose you assign to it. This device uses a non-standard keycode set (Keycode Set 2) which is shown below.

### Keycode Set 2 for the Function Box

(press value/release value)

	0/1	2/3	4/5	6/7	
8/9	10/11	12/13	14/15	16/17	18/19
20/21	22/23	24/25	26/27	28/29	30/31
32/33	34/35	36/37	38/39	40/41	42/43
44/45	46/47	48/49	50/51	52/53	54/55
	56/57	58/59	60/61	62/63	

The HP 46086A (Function Box) responds to the following HP-HIL commands when sent by the HIL SEND statement:

- PRM
- ACK
- DKA
- EKA 1
- EKA 2



The **HP 46030A (Vectra Keyboard)** is not supported on BASIC/UX.

The **HP 92916A (Bar-Code Reader)** reads all standard bar-codes using a wand as the input device. It provides you with an effective and reliable alternative to the time consuming keyboard for data entry. Note that BASIC supports this device in both the ASCII transmit mode, where the input from the device is ASCII characters and in the Keyboard mode<sup>1</sup>, where it transmits the same keycodes as an HP 46020/21A Keyboard. The codes which can be read by the Bar-Code Reader are: 3 of 9, Interleaved 2 out of 5, UPC/EAN, and Codabars USD-4 and ABC.

When the HP 92916A (Bar-Code Reader) is in the ASCII transmit mode use the following statement:

- ON HIL EXT

When the HP 92916A (Bar-Code Reader) is in the Keyboard mode use the following statements:

- ON KBD
- ENTER KBD
- INPUT
- LINPUT

---

<sup>1</sup> When in the Keyboard mode, this device returns an HP-HIL ID in the same range as an HP 46020/21A Keyboard.

---

## Communicating with HP-HIL Devices

This section of the chapter covers the use HP-HIL devices which support the HIL SEND and ON HIL EXT statements. In the examples covered in this section, you will be looking at four HP-HIL devices and how to use them in the HP-HIL link.

- ID Module
- Function Box
- Touchscreen
- Bar Code Reader

### HP-HIL Device Characteristics

Once the HP-HIL device is in the link, you will need to verify its address and determine its characteristics. Accessing this information is the purpose of this section.

To verify a device's address and determine its characteristics, use the HIL SEND *address*;IDD statement and HILBUF\$ function. The HIL SEND *address*;IDD statement executes the HP-HIL Identify and Describe command. Data resulting from the execution of this command is placed in the buffer used by the HILBUF\$ function. Assuming that the address of your device is 1, entering this program and running it will give you the information you need. Note that the information returned is hexadecimal and will have to be interpreted using the information found in the "HP-HIL Appendix" of this manual.

```
100 HIL SEND 1;IDD
110 A$=HILBUF$
120 FOR I= 1 TO LEN(A$)
130   B$=IVAL$(NUM(A$[I]),16)
140   PRINT B$[3];" ";
150 NEXT I
160 END
```

Results from executing this program can be found under the topic heading "Device Characteristics" in each of these sections:

- ID Module
- Function Box and Vectra Keyboard
- Touchscreen
- Bar Code Reader

## ID Module

This module provides a means for securing your software. In this section, you will be:

- Determining ID Module characteristics,
- Verifying your ID Module's product/exchange and serial numbers,
- Learning how to install and remove the ID Module.

### Device Characteristics

This section provides and explains the results from executing the program found in the section entitled "HP-HIL Device Characteristics". Remember that these results assume your ID Module is located at address 1. The program results are as follows:

00 04 01 34 04

where:

- 00 is a buffer overflow count. Zero means the buffer has not overflowed since last read. If the buffer of the HILBUF\$ function overflowed, this value would represent the number of packets of information lost.
- 04 is the number of bytes of data to follow this byte and including this byte. The number of bytes is 4.
- 01 is the address of the device in the loop. The address of the device in this case is 1 which means that it is the first device in the link with an address.
- 34 is the type of device located at the address given. The device in this case, as interpreted from the "Device ID Byte Definitions" table found in the "HP-HIL Appendix" in this manual, is the ID Module.
- 04 is the Describe Record for the device. This record helps you determine the device characteristics. To interpret this hexadecimal value, you need to turn to the "HP-HIL Appendix" found in this manual. Looking in the section entitled "Describe Record", you will find that if bit 2 is set then the device reports security code information.

## Interpreting ID Module Data

In this section, you will learn how to verify the product/exchange and serial numbers for your ID Module.

To verify your product/exchange number, type in and execute the following program:

```
100  Sn$=SYSTEM$("SERIAL NUMBER")
110  OUTPUT Sn_disp$ USING "9D";256*(256*(256.*(NUM(Sn$[8]) MOD 64)
+NUM(Sn$[7]))+NUM(Sn$[6]))+NUM(Sn$[5])
120  PRINT VAL$(256*(256.*BIT(NUM(Sn$[4]),7)+NUM(Sn$[3]))+NUM(Sn$[2]))&CHR$
(NUM(Sn$[4]) MOD 128),Sn_disp$[1,4]&CHR$(NUM(Sn$[9]) MOD 128)&Sn_disp$[5]
130  END
```

The results from executing the above program look similar to this:

```
46084A    2529A10988
```

The same results can be obtained using the HP-HIL Report Security Code command (RSC) in the above program. This requires replacing program line **100** with three additional program lines as shown below. Note that you may need to replace line **120** with additional statements if your program is also using HILBUF\$ to return other data.

```
100  HIL SEND 3;RSC
110  Temp_sn$=HILBUF$
120  Sn$=Temp_sn$[4,12]
130  OUTPUT Sn_disp$ USING "9D";256*(256*(256.*(NUM(Sn$[8]) MOD 64)
+NUM(Sn$[7]))+NUM(Sn$[6]))+NUM(Sn$[5])
140  PRINT VAL$(256*(256.*BIT(NUM(Sn$[4]),7)+NUM(Sn$[3]))+NUM(Sn$[2]))&CHR$
(NUM(Sn$[4]) MOD 128),Sn_disp$[1,4]&CHR$(NUM(Sn$[9]) MOD 128)&Sn_disp$[5]
150  END
```



### Note about Installing and Removing ID Modules

The HP 46084 (ID Module) is an HP-HIL device which connects to the computer through the HP-HIL (HP Human-interface Link) interface. Normally you will be connecting this module to the computer before booting the system. When the KBD binary is loaded, the system recognizes that the module is installed. The **SYSTEM\$** function reads the module's contents each time the function is accessed, rather than keeping the contents in memory.

The ID Module can also be installed while the computer is running. However, in order for BASIC to recognize that it has been connected, you must execute this statement:

**SCRATCH A**

Executing this statement performs a "re-configuration" of the link, after which the BASIC system recognizes and can properly talk to any additional HP-HIL device.

If your machine has both an ID PROM and an ID Module, the ID Module has precedence. In other words, if both are installed (and recognized at boot or **SCRATCH A**), then the ID Module's contents are read and returned by the **SYSTEM\$** function.

If you remove the ID Module and do not re-boot or execute **SCRATCH A**, then the **SYSTEM\$** function will return a null string (even if an ID PROM is present). This behavior is due to the fact that the system still expects the ID Module to be installed, and thus reads nothing when you attempt to read it with **SYSTEM\$**.

Conversely, if you install an ID Module in a machine with an ID PROM after booting BASIC and **without** performing a **SCRATCH A**, then **SYSTEM\$("SERIAL NUMBER")** will return the ID PROM's contents (because it does not recognize that the ID Module is present).

## Function Box

This HP-HIL device has 32 keys and uses Keycode Set 2 which is found in the section entitled "Other Keyboards and Button Devices." Topics covered in this section are as follows:

- Determining Function Box characteristics
- Activating the Function Box
- Trapping Key Presses
- Assigning Functions to Keys.

### Determining Function Box Characteristics

This section provides and explains the results from executing the program found in the section entitled "HP-HIL Device Characteristics". These results assume your ID Module is located at address 1. The program results are as follows:

00 05 01 30 10 80

where:

- 00 is an overflow indicator. If the buffer to the HILBUF\$ function overflowed, this value would represent the number of packets of information lost.
- 05 is the number of bytes contained in the packet of information sent to the buffer used by the HILBUF\$ function including that byte.
- 01 is the address of the device within the HP-HIL link. The address of the device is 1 in this example.
- 30 is the ID number of the device. This number helps to determine what devices are connected in the HP-HIL link. Hexadecimal 30 is the ID number for the Button Box as found in the table entitled "Device ID Byte Definitions" in the "HP-HIL Appendix."

- 10 is the Describe Record Header. It returns information, such as what type of HP-HIL commands are supported by this device, proximity in/out information, and coordinate information. By use of the Describe Record Header information provided in the "HP-HIL Appendix" you will be able to interpret the information contained in this byte. In this case, the 4th bit of the Describe Record byte is set which indicates that the last byte in the packet of information is the I/O Descriptor Byte.
- 80 is the I/O Descriptor Byte. This byte contains information as found in the "I/O Descriptor Byte" table in the "HP-HIL Appendix." You will find that bit 7 of this byte has been set. This indicates that the HP-HIL General Prompt and Acknowledge are supported by this device.

### Activating the Function Box

A status light is located in the upper-right corner of your Function Box. You could use this light to indicate whether the buttons on the Function Box are active or non-active. The following program which is entitled "Activate" can be found in the directory called `/usr/lib/rmb/demo`. Note that the program assumes your Function Box's address is 2. You may have to change this address if your Function Box's address is different from that found in the program.

```
100 CLEAR SCREEN
110 DISP "Do you want to activate the Function Box?";
120 DISP " Enter Yes or No.";
130 INPUT "",Response$
140 IF LWC$(Response$[1,1])="y" THEN
150     HIL SEND 3;PRM
160     ON HIL EXT 8 CALL Key_service
170     PRINT TABXY(15,10),"The status light is on and keys are active."
180 ELSE
190     HIL SEND 3;ACK
200     OFF HIL EXT
210     PRINT TABXY(15,10),"The status light is off and keys are not active."
220 END IF
230 Loop: GOTO Loop
240 END
250 !
260 SUB Key_service
270     PRINT "Key_service called."
280 SUBEND
```

This program executes the HP-HIL General Prompt and Acknowledge commands using the statements found on lines **150** and **190** of the above program. When the program is run you are prompt by the following message:

Do you want to activate the Function Box? Enter Yes or No.

You need to type in either **Yes** or **No**. If your answer is **Yes**, the status light on the Function Box lights up and this message is displayed:

The status light is on and keys are active.

Each time a button on the Function Box is pressed or released, this message is displayed:

Key\_service called.

Note that the subprogram called **Key\_service** just prints a message that it has been called. It is left up to you to write your own subprograms to assign processes to the keys. If you answered **No** to the prompt, the status light either remains off if it was already off or is turned off if it was on and the following message is displayed:

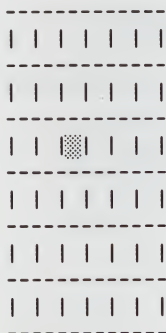
The status light is off and keys are not active.


Press the **[PAUSE]** key which pauses the program and allows you to re-run it or to go on to the next example.



## Trapping Key Presses

Key presses are recognized as interrupts and cause end-of-line branching when the ON HIL EXT statement is executed in your program. Recognition of a key press and then branching to a subprogram is called “trapping” a key press. The program given in this section provides a good example of trapping key presses. This program is called “Mul\_press” and can be found in the directory called `/usr/lib/rmb/demo`. When you enter and run this program a Function Box key matrix is displayed on the screen. Each time you press a key that key’s location in the key matrix is displayed on the screen. Note that you should only press one key at a time because the Function Box **does not** provide for multi-key presses. For example, pressing key number 12 on the Function Box results in the following being displayed:



```
-----  
| | | | | | |  
-----  
| | | | | | |  
-----  
| |  | | | | |  
-----  
| | | | | | |  
-----  
| | | | | | |  
-----  
| | | | | | |  
-----
```

Releasing the same key you pressed causes key 12's matrix location to go blank. The following program entitled "Mul\_press" produced the above results. Note that this program assumes a Function Box address of 3. If your Function Box is not located at address 3, then you need to change the addresses on both lines **120** and **290** of the program to the proper address for your Function Box.

```

100  CLEAR SCREEN
110  !               Assumes button box at location 3.
120  HIL SEND 3;DKA ! Disable Key Auto-repeat.
130  !
140  COM INTEGER Array(31,1:2)
150  INTEGER Key,Packet_length,Packet_start,Index,Packet_end
160  INTEGER Keycode
170  DIM A$[256]
180  !
190  DATA 2,1,3,1,4,1,5,1
200  DATA 1,2,2,2,3,2,4,2,5,2,6,2
210  DATA 1,3,2,3,3,3,4,3,5,3,6,3
220  DATA 1,4,2,4,3,4,4,4,5,4,6,4
230  DATA 1,5,2,5,3,5,4,5,5,5,6,5
240  DATA 2,6,3,6,4,6,5,6
250  !
260  READ Array(*)
270  Framework
280  !               enable device at location 3 (button box)
290  ON HIL EXT 2^3 GOSUB Service_req
300  !
310  Loop:GOTO Loop
320  !
330  Service_req:!
340  A$=HILBUF$
350  IF LEN(A$)=1 THEN RETURN ! no data in buffer
360  Packet_start=2
370  REPEAT
380    Packet_length=NUM(A$[Packet_start])
390    Packet_end=Packet_start+Packet_length-1
400    FOR Index=Packet_start+3 TO Packet_end
410      Keycode=NUM(A$[Index])
420      Key=(Keycode DIV 2)
430      Disp_key(Key,NOT BIT(Keycode,0))
440    NEXT Index
450    Packet_start=Packet_end+1
460  UNTIL Packet_start>LEN(A$)
470  RETURN
480  !
490  END

```

```

500 SUB Disp_key(INTEGER N,On)
510   COM INTEGER Array(*)
520   IF On THEN                                ! Even='downstroke'.
530     PRINT TABXY(2*Array(N,1)+17,2*Array(N,2)+4),"■";
540   ELSE !                                     Odd=>'upstroke'.
550     PRINT TABXY(2*Array(N,1)+17,2*Array(N,2)+4)," " ;
560   END IF
570 SUBEND
580 SUB Framework
590   FOR I=0 TO 10 STEP 2
600     PRINT TABXY(18,I+5);"-----"
610     PRINT TABXY(18,I+6);"| | | | | |"
620   NEXT I
630   PRINT TABXY(18,17);"-----"
640 SUBEND

```

Here is an explanation of the above program.

Line **100** clears the display.

Line **120** disables the auto-keyswitch repeat mode by executing the HP-HIL command DKA.

Lines **140** through **170** declare the variables for the program.

Lines **190** through **240** provide values for the element locations in the two dimensional array called **Array**. Note that **OPTION BASE 0** is used for this program.

Line **260** is a **READ** statement which assigns all of the values in the **DATA** statements of lines **190** to **240** to the elements in the array called **Array**.

Line **270** calls the subprogram **Framework** which causes a 6 by 6 Function Box key matrix to be displayed on the screen. The subprogram **Framework** consist of lines **580** to **640**.

Line **290** enables end-of-line branching when a key on the Function Box is pressed.

Line **310** is a continuous loop which allows the program to wait for key presses.

Line **330** is the label for the beginning of the service routine called **Service\_req**.

Line **340** assigns the value of the buffer used by the function **HILBUF\$** to the string array called **A\$**.



Line **350** tests the string length. If the string length is 1, then the data that generated this interrupt has already been read and a return from the subroutine is made.

Line **360** assigns the value of 2 to the integer variable `Packet_start`. Note that `Packet_start` initially is the subscript for the second element in `A$`. This element tells how many elements there are in the first packet of information including that element. This variable will also be the counter used to determine the starting position of each packet of information in the string (`A$`).

Lines **370** through **460** are a REPEAT loop used to search the data in the string (`A$`) for each packet of key press information. Lines **400** through **440** scan the packet for the up or down key presses. Line **430** detects the up or down key press and passes this parameter to the subprogram called `Disp_key`. Note that the integer variable `Key` is the key which was either pressed or released.

Line **470** is the return back from the subroutine.

Lines **500** through **570** are the subprogram called `Disp_key`. This subprogram has a test in it for an up or down press of a key on the Function Box. Each time you press a key that key's location in the key matrix is displayed on the screen as an inverse video character. When you release that key a blank appears in the key matrix.

Lines **580** through **640** are the subprogram called `Framework` which draws the key matrix on the display.

### **Assigning Functions to Keys**

It was previously mentioned that processes or functions can be assigned to each key on the Function Box. These functions are not assigned in the same manner as those assigned to typing-aids keys nor do they have softkey labels which appear at the bottom of the display.

A function is assigned by pressing a key which causes an interrupt. This interrupt is trapped and causes a branch to a subprogram which sets a process in motion. Once the process is completed the subprogram returns execution back to the main program and waits for another key press. An example of this can be seen by `LOADing` and executing the following program called "Button\_box" can be found in the directory called `/usr/lib/rmb/demo`. Keep in mind that only two keys are being used in this program. These keys are located in the top row starting from the left. Pressing the first key stops the program, pressing the second key draws a series of circles. Any other key press causes the following message to appear on the display:

This key is not implemented.



Note that you must have the graphics binary (GRAPH) loaded in order for this program to work. Also, lines **120**, **130**, and **400** may have to be changed if your Function Box is not located at address 3 in order for the program to work.

```
100  INTEGER Packet_length,Packet_start,Packet_end
110  DIM A$[256]
120  ON HIL EXT 2^3 GOSUB Service_req
130  HIL SEND 3;PRM
140  CLEAR SCREEN
150  GINIT
160  PEN 0
170  GRAPHICS ON
180  Loop:GOTO Loop
190  !
200  Service_req: !
210  A$=HILBUF$
220  IF LEN(A$)=1 THEN RETURN
230  Packet_start=2
240  REPEAT
250    Packet_length=NUM(A$[Packet_start])
260    Packet_end=Packet_start+Packet_length-1
270    FOR Index=Packet_start+3 TO Packet_end
280      Key_check(NUM(A$[Index]))
290    NEXT Index
300    Packet_start=Packet_end+1
310  UNTIL Packet_start>LEN(A$)
320  RETURN
330  !
340  Prog_done:END
350  !
360  SUB Key_check(INTEGER Key_num)
370    SELECT Key_num
380      CASE 0,1
390        DISP "The program has STOPPED!"
400        HIL SEND 3;ACK
410        STOP
420      CASE 2
430        MOVE 50,50
440        FOR I=1 TO 20
450          POLYGON I,20,20
460        NEXT I
470        FOR I=20 TO 1 STEP -1
480          POLYGON I,20,20
490        NEXT I
500        GCLEAR
```

```

510     CASE ELSE
520         IF (Key_num MOD 2)=0 THEN
530             PRINT TABXY(20,10),"This key is not implemented."
540             WAIT 1
550             CLEAR SCREEN
560         END IF
570     END SELECT
580 SUBEND

```

The following is an explanation of the above program. This program assumes your Function Box is located at address 3 in the HP-HIL link.

Lines **100** and **110** declare the integer and string variables.

Line **120** executes the statement `ON HIL EXT 8` which sets up a branch to be made to the subprogram `Service_req`. At the same time this branch is initiated Poll Record data is sent to the buffer used by the function `HILBUF$`. This data contains information on which key was pressed. You can trap these key presses and use them to activate various process.

Line **130** turns on the status light of the Function Box.

Line **140** clears the alpha display.

Line **150** set the graphics parameters to their default values.

Line **160** sets the graphics pen value to zero (0).

Line **170** turns the graphics display on.

Line **180** causes the program to loop until a key is pressed.

Line **200** is the label for the beginning of the service routine called `Service_req`.

Line **210** assigns the value of the buffer used by the function `HILBUF$` to the string called `A$`.

Line **220** tests the string length. If the string length is 1, then the data associated with this interrupt has already been processed, and a return from the subroutine is made.

Line **230** assigns the value of 2 to the integer variable **Packet\_start**. Note that **Packet\_start** initially is the subscript for the second element in **A\$**. This element tells how many elements there are in the first packet of information including that element. This variable will also be the counter used to determine the starting position of each packet of information in the string (**A\$**).

Lines **240** through **310** are a REPEAT loop used to search the data in the string (**A\$**) for each packet of key press information. Lines **270** through **290** search the packet for up or down key presses. Line **280** calls the subprogram **Key\_check** and passes it the value of the key you have pressed.

Line **320** is the return back from the subroutine.

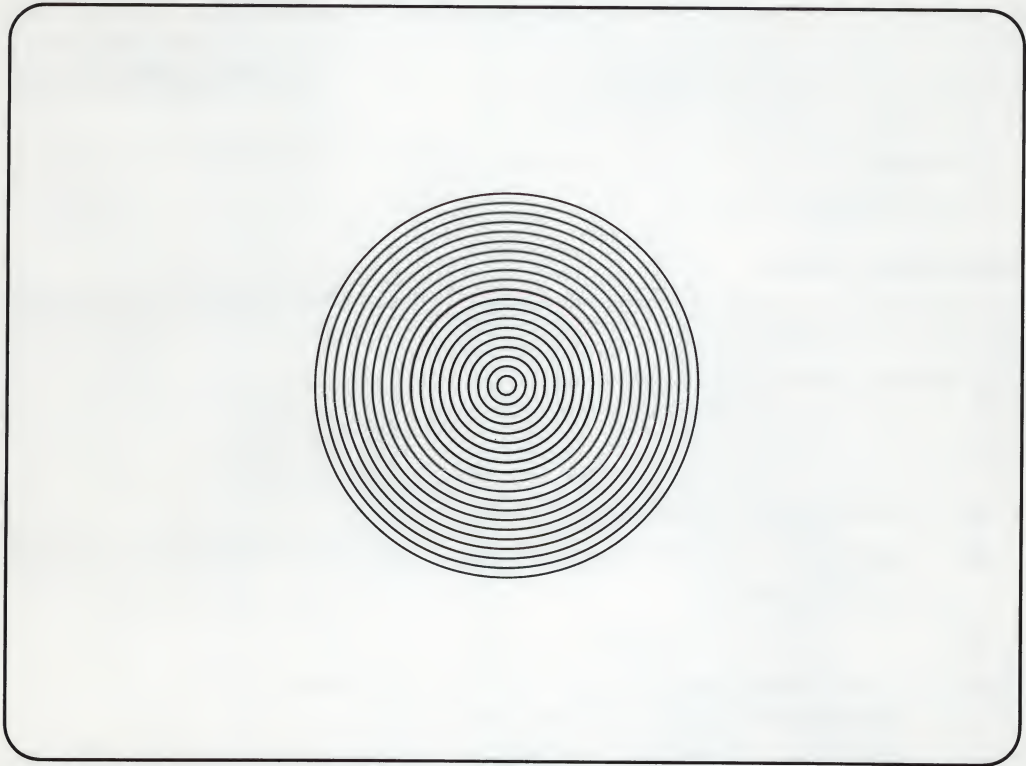
Lines **360** through **580** are the subprogram called **Key\_check**. This subprogram is a large **SELECT** structure starting at line **370** and going to line **570**. This structure selects a particular process to be performed depending on which key has been pressed. One process can be found in each of the three different **CASE** segments.

Lines **380** through **410** are the first **CASE** segment. This segment when executed causes the following message to be displayed:

**The program has STOPPED!**

It also executes the HP-HIL command **ACK** (General Acknowledge) which turns the status light on the Function Box off and terminates the program.

Lines **420** through **500** are the second **CASE** segment. This segment causes circles to be displayed one inside the other starting with a small circle and going to a large one. It then erases these circles in the reverse order.



Lines **510** to **560** are the third **CASE** segment. All key releases come through this **CASE** segment and are ignored due to line **520**. This includes the release of key 2. Any key press coming here causes the following message to be displayed:

**This button is not implemented.**

Remember there are only two keys whose interrupts were recognized as a result of running this program. When you press one of the keys which does not cause a process to become activated the above message is displayed.



## Using a Touchscreen

As its name indicates, the Touchscreen responds to a touch of the screen. A touch of the screen will report you are in proximity and a release of this touch will report you are out of proximity. At the same time this device is reporting in and out proximity information it is also returning X and Y axis coordinate information for the screen touch. Combining both of these characteristics, the user is able to do location selection using the Touchscreen. Below is a list of the topics covered in this section:

- Determining Touchscreen characteristics,
- Plotting selected locations.

### Device Characteristics

Assuming the Touchscreen is located at address 2, it will return Identify and Describe information as follows:

00 0B 02 8C 52 0A 01 38 00 2A 00 08

where:

- 00 is the overflow counter.
- 0B indicates the number of bytes of data to follow including this byte (in this case there are 11).
- 02 is the address of the device.
- 8C is the device type. In this case, it is the Touchscreen as determined from the table entitled, "Device ID Byte Definitions."
- 52 is the Describe Record. This gives information about the device. The bit pattern for a Describe Record of 52 is as follows:
  - Bit 0** is not set and **bit 1** is. This says the device will return X and Y coordinates.
  - Bit 4** is set. This indicates that the last byte of the Describe Record is the I/O Descriptor byte.
  - Bit 5** is not set. This indicates that the X and Y coordinates returned will only be 8 bits each (one byte).
  - Bit 6** is set. This indicates that Absolute Positional data will be returned by the device.

- 0A     These two bytes are combined to give 010A (i.e. the 2nd byte is the more
- 01     significant part of the number). Since bit 5 of the Describe Record is not set, this value is the number of counts per meter (in this case 266).
  
- 38     These two bytes are combined to give 0038. (i.e. the 2nd byte is the more
- 00     significant part of the number). This value represents the total number of absolute graphics units in the X-axis (in this case 56).
  
- 2A     These two bytes are combined to give 002A. (i.e. the 2nd byte is the more
- 00     significant part of the number). This value represents the total number of absolute graphics units in the Y-axis (in this case 42).
  
- 08     is the I/O Descriptor Byte. Bit 3 of this byte is set indicating that the device indicates changes in proximity in or out status in its Poll Record (only returned when the status changes).

### **Plotting Selected Locations**

This task requires the use of the statement `ON HIL EXT`. Information returned by the Touchscreen can be found in the buffer used by the `HILBUF$` function.

The following program called "Touch\_plot", found in the directory called `/usr/lib/rmb/demo`, continuously displays the X and Y coordinates of your finger or stylus as you move it across the screen. The first release of your touch on the screen will cause a `MOVE` to that position. Any subsequent screen releases will cause a line to be draw from the last coordinate position to the present one. This particular program only allows you to plot and draw lines to 6 different locations on the screen.

Below are some sample results which you could receive from entering and running the program in this section.



Point 6 : X=40 Y=40

The following program called "Touch\_plot" returned the above results. Note that in order for this program to work on your BASIC system you need to change the address on line **210** to the address of your Touchscreen. The address currently assigned to this HP-HIL device in the program is 1.

```
100 CLEAR SCREEN ! Clear alpha.
110 GINIT        ! Initializes graphics.
120 GRAPHICS ON  ! Turn on graphics.
130 WINDOW 0,56,0,43 ! Scale to match Touchscreen resolution.
140 !
```

```

150  INTEGER Test,Point,Packet_start,Packet_length,Packet_end
160  INTEGER In_proximity,X_coord,Y_coord
170  DIM A$[256]
180  !
190  PRINT TABXY(16,12),"Touch the screen at 6 different locations."
200  !
210  ON HIL EXT 2 GOSUB Service_req ! Assumes the Touchscreen is the
220                                ! first device on the link.
230  Point=1
240  !
250  Loop:GOTO Loop
260  !
270  Service_req:  !
280  IF Point=1 THEN CLEAR SCREEN
290  A$=HILBUF$
300  IF LEN(A$)=1 THEN RETURN
310  Packet_start=2
320  REPEAT
330    Packet_length=NUM(A$[Packet_start])
340    Packet_end=Packet_start+Packet_length-1
350    IF BIT(NUM(A$[Packet_start+2]),1)=1 THEN
360      X_coord=NUM(A$[Packet_start+3])
370      Y_coord=NUM(A$[Packet_start+4])
380      DISP "Point ";Point;" : X = ";X_coord;" Y = ";Y_coord
390    END IF
400    IF BIT(NUM(A$[Packet_start+2]),6)=1 THEN
410      In_proximity=NUM(A$[Packet_end])
420      IF In_proximity=142 THEN
430        IF Point=1 THEN
440          MOVE X_coord,Y_coord
450        ELSE ! Point=2 thru 6.
460          DRAW X_coord,Y_coord
470        END IF
480      END IF
490    END IF
500    Packet_start=Packet_end+1
510  UNTIL Packet_start>LEN(A$)
520  IF In_proximity=143 THEN
530    Point=Point+1
540  END IF
550  IF Point<7 THEN RETURN
560  DISP "You're Done"
570  !
580  END

```



The following is an explanation of the above program.

Line **100** clears the alpha screen.

Line **110** initializes graphics to its default values.

Line **120** turns graphics on.

Line **130** sets the graphics scale to match the Touchscreen resolution.

Lines **150** through **170** declare the program variables.

Line **190** prompts the user to make 6 screen touches. This means moving your finger or stylus in and out of proximity 6 times.

Line **210** enables end-of-line interrupts from the Touchscreen located at address 1.

Line **230** initializes the counter variable to 1. The counter variable is called **Point** and it keeps track of the number of times you have released your finger or stylus from the screen.

Line **250** is a continuous loop which allows the system to wait for either a screen touch or release.

Lines **270** through **560** is a subroutine called **Service\_req**. Whenever a touch or release of the screen is made this service routine is called.

Line **290** assigns the value of the buffer used by the function **HILBUF\$** to the string called **A\$**.

Line **300** tests the string length. If the string length is 1, then the data associated with this interrupt has already been processed, and a return from the subroutine is made.

Line **310** assigns the value of 2 to the integer variable **Packet\_start**. Note that **Packet\_start** initially is the subscript for the second element in **A\$**. This element tells how many elements there are in the first packet of information including that element. This variable will also be the counter used to determine the starting position of each packet of information in the string (**A\$**).

Lines **320** through **510** are a REPEAT loop used to search the data in the string (**A\$**) for each packet of screen touch and release information. Lines **350** through **490** check the packet for coordinate and proximity information. The REPEAT loop continues until the last element in the string **A\$** is reached.

Lines **400** through **480** test for proximity in and out. As long as proximity in is detected the coordinates of your present finger or stylus position are printed. Lines **430** through **470** determine whether to draw a line on the display or to move the graphics pen to the initial position before plotting.

Line **560** is reached when the sixth point is plotted on the screen. This line will cause the following to be displayed:

**You're Done**

## **Using a Bar Code Reader**

A Bar Code Reader may either act as a keyboard or a transmitter of ASCII characters. In this section, you will assume it is a transmitter of ASCII characters. When your Bar Code Reader is acting as a keyboard it is returning keyboard presses. When it is acting as an ASCII transmitter it is sending ASCII characters.

To use this HP-HIL device as a reader of ASCII characters you need to program the switches on its underside for the proper settings. The settings for these keys are explained in the installation manual for this device. Below is a list of settings you need to verify on the Bar Code Reader before booting the system.

- The four switches used to define the Transmission Type (i.e. switches 5 through 8 on the right-hand set of switches) should be set to all zeros. This puts you in the non-keyboard mode.
- The Appended Key switch setting (i.e. switches 2 and 3 on the right-hand set of switches) should be set for none. This assures that no key operation will be appended to the end of your bar-code reading.

- The Bar Code Reader should have its Auto Recognition switch set (i.e. switch 1 on the right-hand set of switches) and the bar code you are going to be reading selected (use the eight left-hand set of switches). The following are possible bar code selections:

- Interleaved 2/5
- Code 3/9
- Extended Code 3/9
- CODABAR USD-4 and ABC
- UPC/EAN/JAN
- UPC E (8 digits)

Note that the Automatic code recognition does not mean that the bar-code reader will automatically know the codes you intend to read, you have to select them first. It does mean that it will automatically recognize the codes you have selected. For example, if you wanted to read bar codes that may be either the Interleaved 2 of 5 bar code or the 3 of 9 bar code, you would set the right-hand set of switches to the following:

- switch 8 to 1
- switch 7 to 1
- switches 6 through 1 to 0

Topics covered in this section are:

- Determining device characteristics, and
- Transmitting ASCII Characters.

### **Determining Bar Code Reader Characteristics**

The following results assume the Bar Code Reader is at address 4. The Bar Code Reader returns Identify and Describe information as follows:

00 04 04 5C 00

If this is not the case, you need to check the switch settings on the underside of your Bar Code reader again.



After the system has been re-booted you should do another Identify and Describe of the device to see that it is recognized as a Bar Code Reader. The program used to obtain this information is found in the section entitled "HP-HIL Device Characteristics." Your results after entering and running this program should be as follows:

```
00 04 04 5C 00
```

where:

- 00     is the null character (packet overflow count).
- 04     is the number of bytes to follow including this byte.
- 04     is the address of the device.
- 5C     is the type of device which in this case is the Bar Code Reader.
- 00     is the Describe Record Header with no special features.

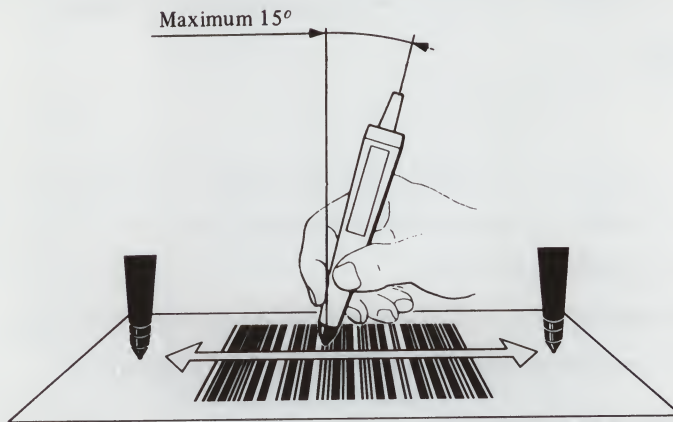
### Reading a Selected Bar Code

Once you have selected the type of bar code you wish to read, you are ready to use your bar code reader. To do this, enter the program called "Bar\_code" found in the directory called `/usr/lib/rmb/demo` and run it. Note that if your Bar Code Reader is not located at address 4, then you need to change line **100** of the program to match your devices address.

```
100  ON HIL EXT 2~4 CALL Disp_buf
110  Loop:GOTO Loop
120  END
130  !
140  SUB Disp_buf
150      DIM A$[256]
160      A$=HILBUF$
170      IF LEN(A$)=1 THEN RETURN
180      Packet_start=2
190      REPEAT
200          Packet_length=NUM(A$[Packet_start])
210          Packet_end=Packet_start+Packet_length-1
220          PRINT A$[Packet_start+3,Packet_end];
230          Packet_start=Packet_end+1
240      UNTIL Packet_start>LEN(A$)
250      PRINT
260  SUBEND
```



To have data displayed on the screen, you need to move the Bar Code Reader's wand rapidly and at a constant speed across the bar code. The wand should also be held as shown:



**Figure 16-3. The Correct Method for Holding the Bar Code Reader**

The following is an explanation of the program provided in this section.

Line **100** enables end-of-line branching on movement of the wand across the bar code.

Line **110** is a continuous loop which allows the program to idle while waiting for a bar-code reading.

Line **140** is the beginning of the subprogram `Disp_buf`. This subprogram is used to display the data read by the Bar Code Reader.

Line **160** dimensions the string `A$`.

Line **170** tests for an empty buffer. If the buffer is empty the data that caused the interrupt has already been processed by a previous invocation of the service routine, so it simply returns to the idle loop on line **110**.

If the buffer is not empty, a REPEAT loop in lines **190** through **240** causes ASCII bar code information to be displayed.

Line **180** assigns the value of 2 to the integer variable `Packet_start`. Note that `Packet_start` initially is the subscript for the second element in `A$`. This element tells how many elements there are in the first packet of information including that element. This variable will also be the counter used to determine the starting position of each packet of information in the string (`A$`).

Lines **190** through **240** are a REPEAT loop used to search the data in the string (`A$`) for each packet of information in the string `A$`.

Line **220** prints out the ASCII characters found in each packet.

To end the program press the Stop key.

## Interaction Among Multiple HP-HIL Devices

End-of-line interrupts can be handled when they come from more than one HP-HIL device during program execution. To demonstrate this a program called `Multi_dev`, found in the directory `/usr/lib/rmb/demo`, has been provided for you to load and run. The program and its explanation are included in this section. Note that line **1030** of this program assumes that you have a Touchscreen located at address 1 and a Function Box located at address 3. If these are not the correct addresses for your devices, you will have to change the address mask on line **1030** of this program. For information on how to change the address mask, read the section in this chapter entitled "Communicating through the HP-HIL Interface."

The interaction covered in this section is between a Touchscreen and a Function Box. Both of these devices could easily be replaced by two other HP-HIL devices which are supported by the ON HIL EXT and/or HIL SEND statements. To do this, you would have to make a few variables changes to suit the new program and write new subprograms which would be appropriate for the HP-HIL devices you are using.

```
1000 DIM Buf$(256),Packet$(15)
1010 INTEGER Index,Hil_addr
1020 !
1030 ON HIL EXT 10 GOSUB Disp_buf ! Set up interrupts for
1040 !                               addresses 1 and 3.
1050 GINIT ! Initialize
1060 GRAPHICS ON ! Turn on graphics.
1070 PRINT TABXY(16,4)," This program allows you to touch a point on"
1080 PRINT TAB(16),"the screen and draw a figure at that location"
1090 PRINT TAB(16),"by pressing a key on the Function Box. The"
1100 PRINT TAB(16),"keys are numbered from left to right starting"
1110 PRINT TAB(16),"with the top row of Function Box keys."
```

```

1120 PRINT
1130 PRINT TAB(21),"Key 1 draws a TRIANGLE."
1140 PRINT TAB(21),"Key 2 draws a SQUARE."
1150 PRINT TAB(21),"Key 3 draws a PENTAGON."
1160 PRINT TAB(21),"Key 4 draws a CIRCLE."
1170 PRINT
1180 PRINT TAB(16),"To continue with the program:"
1190 PRINT
1200 PRINT TAB(21),"Press 'Continue', or"
1210 PRINT TAB(21),"Type 'CONT' and press 'Return'."
1220 PAUSE
1230 CLEAR SCREEN ! Clear the alpha display.
1240 WINDOW 0,56,0,43 ! Scale to match Touchscreen resolution.
1250 !
1260 Loop:GOTO Loop
1270 !
1280 Disp_buf: !
1290 Buf$=HILBUF$
1300 IF LEN(Buf$)=1 THEN RETURN ! Data already processed.
1310 Packet_start=2 ! Skip "overflow" indicator.
1320 REPEAT
1330   Packet_length=NUM(Buf$[Packet_start]) ! Determine packet length.
1340   Packet_end=Packet_start+Packet_length-1 ! Find end of packet.
1350   Packet$=Buf$[Packet_start,Packet_end]
1360   !
1370   Hil_addr=NUM(Packet$[2])
1380   SELECT Hil_addr
1390   CASE 1 ! Touchscreen
1400     CALL Touchscreen(Packet$)
1410   CASE 3 ! Function box
1420     CALL Function_box(Packet$[4])
1430   END SELECT
1440   !
1450   Packet_start=Packet_end+1 ! Prepare for next packet.
1460   !
1470 UNTIL Packet_start>LEN(Buf$)
1480 !
1490 RETURN
1500 END
1510 !
1520 SUB Touchscreen(Coordinate$)
1530 !
1540   IF BIT(NUM(Coordinate$[3]),1)=1 THEN
1550     X_coord=NUM(Coordinate$[4])
1560     Y_coord=NUM(Coordinate$[5])
1570     MOVE X_coord,Y_coord
1580   END IF
1590 SUBEND
1600 !

```



```

1610 SUB Function_box(Key_press$)
1620   INTEGER Key_num
1630   WHILE LEN(Key_press$)
1640     Key_num=NUM(Key_press$)
1650     SELECT Key_num
1660       CASE 0,1
1670         POLYGON 5,3,3
1680       CASE 2,3
1690         POLYGON 5,4,4
1700       CASE 4,5
1710         POLYGON 5,5,5
1720       CASE 6,7
1730         POLYGON 5,50,50
1740       CASE ELSE
1750         BEEP
1760     END SELECT
1770     Key_press$=Key_press$[2]
1780   END WHILE
1790 SUBEND

```

The following is an explanation of the above program. This program as lines **1070** to **1110** state allows you to touch a point on the screen and draw a figure at that location by pressing a key on the Function Box. The keys are numbered from left to right starting with the top row of Function Box keys.

Line **1030** initiates the end-of-line interrupts for the HP-HIL devices located at addresses 1 and 3. To do this you need to know how to set up the mask which will cause end-of-line interrupts to be recognized by both devices. The mask value is obtained by raising 2 by the power of each of the addresses and adding these values. For example, 2 raised to the first power added to 2 raised to the third power results in the value 10 for your mask. When an interrupt is received from either of the HP-HIL devices, program execution branches to the subroutine called `Disp_buf`.

The subroutine `Disp_buf` includes lines **1280** through **1490**. This subroutine separates the packets of data sent by each HP-HIL device to the string buffer of the function `HILBUF$` and sends those packets to appropriate subprograms which process this data. In other words, packets containing the address 1 are sent to the subprogram called `Touchscreen` and those packets with address 3 are sent to the subprogram called `Function_box`. Once the string buffer has been completely processed the subroutine is exited.



**Touchscreen** is the subprogram located in lines **1520** through **1590** which searches the string **Coordinates\$** to determine if it has X and Y axis coordinate information. If coordinate information is available, it is assigned to the variables **X\_coord** and **Y\_coord**. The graphics pen is next moved to the location of these coordinates. Plotting of the figures will start at these locations. Once the pen move is made program execution is returned to the main program.

Now that a screen location has been selected pressing a key on the Function Box will cause a triangle, square, pentagon or circle to be drawn at that location. The subprogram called **Function\_box** located at lines **1610** to **1790** receives the string called **Key\_press\$** and looks at it for the number of the key which was pressed and assigns that value to the variable called **Key\_num**. The **SELECT** structure uses the variable **Key\_num** to choose which figure should be drawn at the last selected screen location. Note that the **CASE** segment will responded to both the press and release of a Function Box key. The **WHILE** loop on lines **1630** through **1780** handles multiple keys in the packet, since the program only expects keycodes from the Function Box.

### Modifying the Interactive Program

The program explained in the previous section made the assumption that there was a Touchscreen located at address 1 and a Function Box located at address 3. If you didn't have those HP-HIL devices or they weren't located at the addresses given above, you would need a way of determining what devices were on your HP-HIL link and their address. This section provides a method for doing this.

Determining which HP-HIL devices are on the HP-HIL link and their address, can be accomplished by adding the following **FOR** loop to the your program:

```
1030    ON ERROR GOTO Link_end
1040    FOR I=1 TO 7
1050        HIL SEND I;IDD
1060        Buf$=HILBUF$
1070        Idd(I)=NUM(Buf$[4])
1080    NEXT I
1090    Link_end: ! OFF ERROR
```

These program lines can be inserted in the previous program just after line **1020**. The **FOR** loop consisting of lines **1040** through **1080** is designed to loop **7** times because that is the maximum number of addressable devices you may have on the HP-HIL link at any time. If there are less than **7** devices on the link an error occurs and the **FOR** loop exits to line **1090** labeled **Link\_end**. This branch to the label **Link\_end** is a result of the **ON ERROR** statement on line **1030**.

Line **1050** uses the **HIL SEND** statement along with the **HP-HIL IDD** command to determine the device's location in the HP-HIL link, as well as its Device ID. Using the Device ID number returned upon executing the **HIL SEND address;IDD** statement, you can determine what your device is by looking the number up in the "Device ID Byte Definition" table found in the "HP-HIL Appendix" in the back of this manual.

Information returned after executing the **HIL SEND address;IDD** statement is placed in the string buffer of the **HILBUF\$** function. Line **1060** takes the information found in this string buffer and assigns it to the string variable **Buf\$**.

Line **1070** assigns the integer value of the fourth element of **Buf\$** to the integer array variable **Idd(I)**. The fourth element in **Buf\$** is the Device ID number. "I" (device address) in the subscript portion of the array is incremented as many times as there are devices in the link.

A **SELECT** structure lines **1500** through **1610** can be added to the program to access various subprograms which perform a process for a specified HP-HIL device on the link. A device address called **Hil\_addr** is used as an index to the array **Idd** to obtain the device ID number associated with the index value. For example, if 1 is assign to the variable **Hil\_addr** and **Hil\_addr** is used as the index to the array **Idd** and the device ID number found at that index is 48 (decimal), the subprogram **Function\_box** is called and its process is executed. Note that the additional **SELECT** structure should follow line **1370** of the program. The **SELECT** structure contains the following program lines:

```
1500  SELECT Idd(Hil_addr)
1510  CASE 0 to 31
1520    Vectra(Packet$)
1530  CASE 48
1540    Function_box(Packet$[4])
1550  CASE 92
1560    Bar_code(Packet$)
1570  CASE 140
1580    Touchscreen(Packet$)
1590  CASE ELSE
1600    ! Ignore
1610  END SELECT
```



# Table of Contents

---

## Chapter 17: Using BASIC/UX in the X Window System

Windowing Operations with BASIC/UX .....	17-1
Creating Windows .....	17-2
Listing Windows .....	17-4
Removing Windows .....	17-6
Moving Windows .....	17-7
Outputting Graphics to a Window .....	17-8
Clearing the Contents of Windows .....	17-10
Raising and Lowering a BASIC/UX Window in the Window Stack ....	17-10
Copying Data Between Windows .....	17-12



James M. G. Smith

1870

1871

# Using BASIC/UX in the X Window System

---

# 17

The ability to create windows apart from your BASIC/UX root window allows you to send different types of data to various locations in your windowing environment. For example, one part of your program may send numeric results to one window and graphics drawings to another. This chapter covers the windowing operations that help you do tasks similar to these.

The window keywords covered in this chapter will only work if you are running your BASIC/UX system in the X Window environment. You will also need a mouse and Medium or High Resolution Monitor (e.g., HP 98785A display).

To effectively use BASIC/UX in the X Window System, you should read the chapter “Using BASIC/UX Window Commands” in the *Using BASIC/UX System* manual.

---

## Windowing Operations with BASIC/UX

This section covers the following BASIC/UX windowing operations:

- Creating windows
- Listing windows
- Removing windows
- Moving windows
- Outputting Graphics to a Window
- Clearing the contents of windows
- Raising and lowering a BASIC/UX window in the window stack
- Copying data between windows.

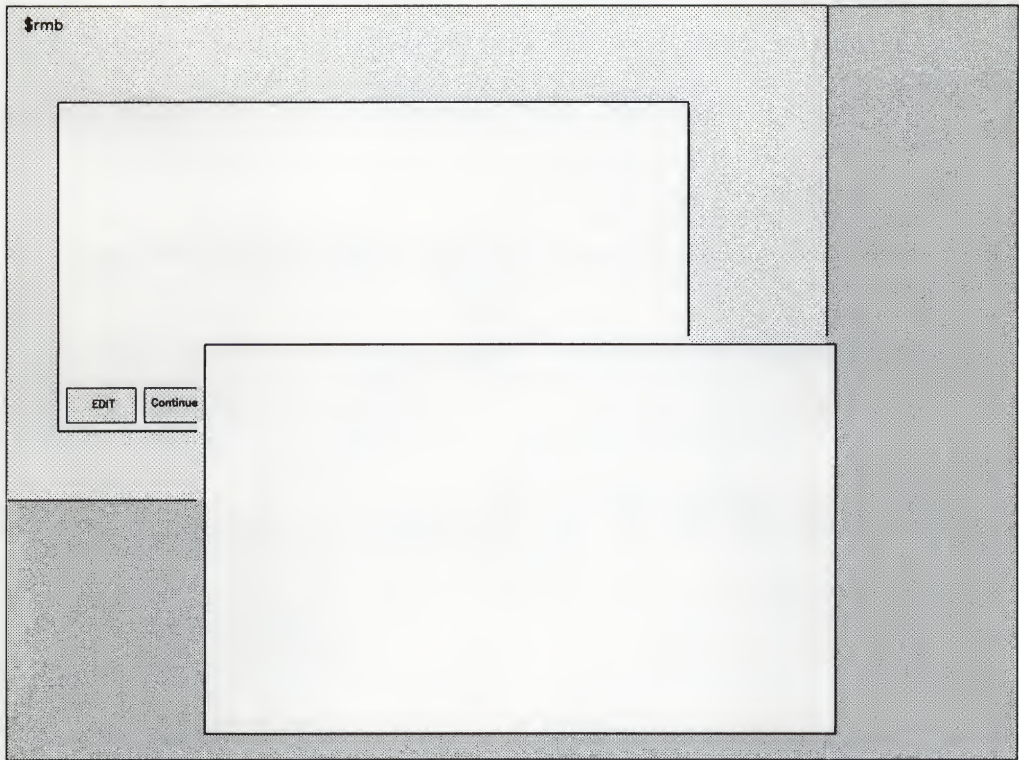
For a detailed explanation of the keywords used in this section, read the *BASIC Language Reference*.

## Creating Windows

A window is a portion of the display that is accessible independently. The ability to access windows as independent printing or plotting devices makes them useful for programs that require the concurrent printing and plotting of numeric and graphics results. You can create one window for your numeric output and another for your graphics output.

The following example program called `create_w` (found in `/usr/lib/rmb/demo`) creates a BASIC/UX window. Lines 110 through 120 assign the coordinates for the upper-left corner of the window being created. These are absolute coordinates from the display's upper-left corner. Lines 130 through 140 assign the window's width and height. The coordinates and window dimensions are used with the `CREATE WINDOW` statement to create window 601. The secondary keyword `LABEL` when used with the `CREATE WINDOW` statement allows you to assign useful names to each window you create.

```
100  INTEGER X_coor,Y_coor,Width,Height
110  X_coor=200  ! Assign the x coordinate position in pixels.
120  Y_coor=350  ! Assign the y coordinate position in pixels.
130  Width=640   ! Assign the width of the window in pixels.
140  Height=400  ! Assign the height of the window in pixels.
150  !
160  ! Create window number 601 and label it as window "One".
170  !
180  CREATE WINDOW 601,X_coor,Y_coor,Width,Height;LABEL "One"
190  END
```



**Figure 17-1. Creating a Window**



## Listing Windows

A listing of the current windows is useful when you need to know a window's attributes. For example, if you needed to know which window numbers have been used or whether the window will retain a graphics image. To list the current windows, execute the following command:

LIST WINDOW

If you executed the program in the previous section called `create_w`, your display will look similar to this:

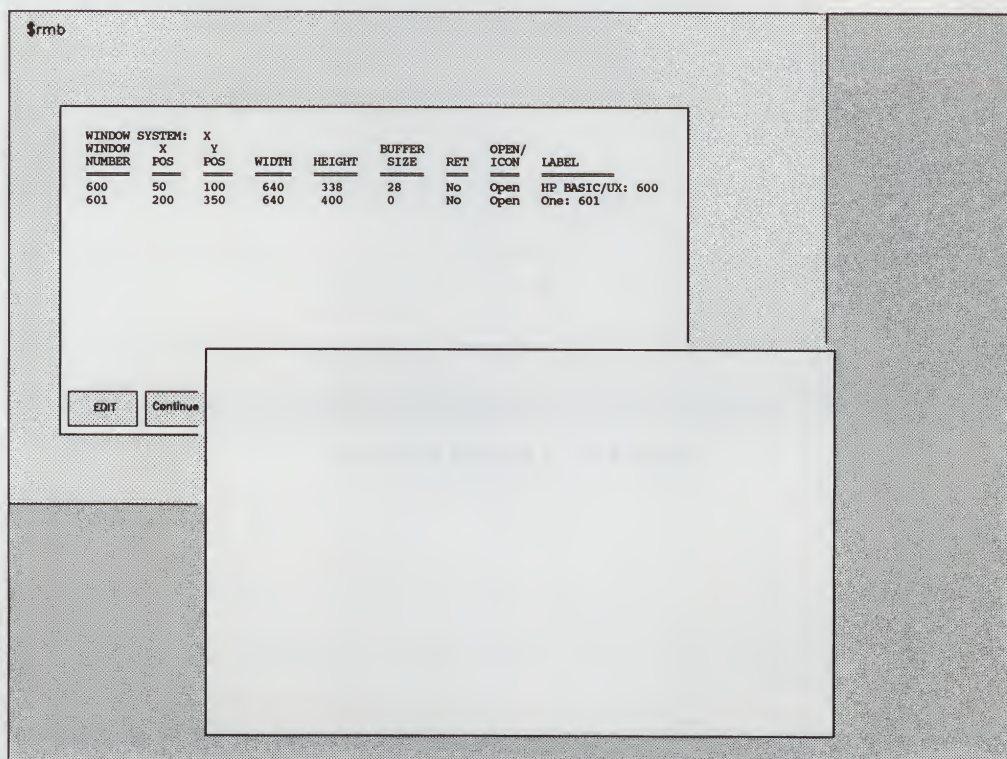


Figure 17-2. Listing of Current Windows and their Attributes

The attributes are explained as follows:

WINDOW NUMBER	gives the window numbers of the currently active windows (e.g., 601).
X POS	gives the x coordinate position in pixels of the upper-left corner of the window listed.
Y POS	gives the y coordinate position in pixels of the upper-left corner of the window listed.
WIDTH	is the window's width in pixels.
HEIGHT	is the window's height in pixels.
BUFFER SIZE	shows the buffer size in lines. Note that lines on the screen can be scrolled into the buffer. These lines are in addition to the current lines in the root window.
RET	indicates with a <b>Yes</b> or a <b>No</b> whether the window will retain a graphics image. This is important if you output graphics information to a window that is obscured by another window and you want to bring it to the top of the "window stack" (for information on the window stack, read the subsequent section "Raising and Lowering a BASIC/UX Window in the Window Stack"). If the window was not retained, the image you would see in the window when you brought it to the top of the window stack would not be a complete one. To retain a window, use the secondary keyword <b>RETAIN</b> with the <b>CREATE WINDOW</b> statement. The secondary keyword <b>RETAIN</b> causes the raster image of graphics in a window to be saved in memory.
OPEN/ICON	shows whether the window you are looking for has been iconified ( <b>Icon</b> ) or is the full window size ( <b>Open</b> ).
LABEL	shows the name you have given to a particular window. This name was assigned when you used the secondary keyword <b>LABEL</b> with the <b>CREATE WINDOW</b> statement.

## Removing Windows

When you are done with a window and its contents, the window can be removed to avoid clutter and confusion on the display. To do this, use either the keyword `DESTROY WINDOW` or `SCRATCH W`. The keyword `DESTROY WINDOW` can be used as a statement in a program or it can be executed from the keyboard line. Note that the keyword `DESTROY WINDOW` allows you to remove one window at a time. The following program called `dest_w` (found in `/usr/lib/rmb/demo`) uses the `DESTROY WINDOW` keyword to remove a window it creates.

```
100  CREATE WINDOW 601,200,250,640,400;LABEL "One"    ! Makes window 601.
110  !
120  WAIT 5
130  DESTROY WINDOW 601                                ! Remove window 601.
140  END
```

You can also remove the window by executing the `SCRATCH W` command. This command differs from the `DESTROY WINDOW` command as follows:

- it is not programmable
- it destroys all created windows.

If you no longer need the BASIC/UX windows that you created using the keyword `CREATE WINDOW` and you want to remove them, execute the following command:

```
SCRATCH W 
```

This command allows you to remove all of your BASIC/UX windows excluding the root window. The `SCRATCH W` command is convenient when you need to remove more than one window.



## Moving Windows

The MOVE WINDOW keyword allows you to move a window to another location on your display. The following program called `move_w` (found in `/usr/lib/rmb/demo`) shows how you can move a window horizontally across your screen using the keyword MOVE WINDOW.

```
100  INTEGER X_coor,Y_coor,Width,Height
110  X_coor=200  ! Assign the x coordinate position in pixels.
120  Y_coor=350  ! Assign the y coordinate position in pixels.
130  Width=640   ! Assign the width of the window in pixels.
140  Height=400  ! Assign the height of the window in pixels.
150  !
160  ! Create window number 601 and label it as window "One".
170  !
180  CREATE WINDOW 601,X_coor,Y_coor,Width,Height;LABEL "One"
190  !
200  ! Move the window horizontally across the screen.
210  !
220  FOR I=1 TO 100 STEP 2
230      MOVE WINDOW 601,X_coor+I,Y_coor
240  NEXT I
250  END
```



## Outputting Graphics to a Window

A window can be assigned as a plotter for your graphics output. For details on plotting, read the section “Using a Plotter with BASIC/UX” in the chapter “Using Printers and Plotters” in this manual. The PLOTTER IS keyword is used to assign a window as a plotting device. The following program called `plot_w` (found in `/usr/lib/rmb/demo`) uses the PLOTTER IS keyword to assign window 601 as the plotting device and draws a rectangle in that window.

```
100  INTEGER X_coor,Y_coor,Width,Height
110  X_coor=200  ! Assign the x coordinate position in pixels.
120  Y_coor=350  ! Assign the y coordinate position in pixels.
130  Width=640   ! Assign the width of the window in pixels.
140  Height=400  ! Assign the height of the window in pixels.
150  !
160  ! Create window number 601 and label it as window "One".
170  !
180  CREATE WINDOW 601,X_coor,Y_coor,Width,Height;LABEL "One"
190  !
200  ! Draw a rectangle in window 601.
210  !
220  GINIT
230  PLOTTER IS 601,"WINDOW" ! Assign window 601 to be the plotter.
240  MOVE 5,5                ! Move the pen to the starting
250                          ! position of the plot.
260  RECTANGLE 60,40         ! Draw the rectangle in window 601.
270  END
```



**Figure 17-3. Sending Graphics Output to a Window**

## Clearing the Contents of Windows

You can clear the contents of a window using the CLEAR WINDOW keyword. This keyword can be used to clear the rectangle from the window created in the previous example. To clear window 601, type:

```
CLEAR WINDOW 601 Return
```

Note that the CLEAR WINDOW keyword is similar to the CLEAR SCREEN keyword; however, if you want to execute the CLEAR SCREEN command to clear a window, that window has to be the current PRINTER IS device. If you want to clear only graphics from a window, use the GCLEAR command while the window is the current PLOTTER IS device.

## Raising and Lowering a BASIC/UX Window in the Window Stack

This section explains how to uncover a selected window and bring it to the top of the “window stack” and how to lower a window to the bottom of the “window stack.” A “window stack” is several “windows” that are layered on top of each other.

CRT Control register 22 gives you a means for moving a window to the top or bottom of the window stack. The commands used to do this are as follows:

```
CONTROL wind_num,22;1    raises a window to the top of the window stack
```

```
CONTROL wind_num,22;0    lowers a window to the bottom of the window stack
```

The variable *wind\_num*, in the above command, is a window number between 601 and 699. The value 1 when sent with the above command causes a designated window to be raised to the top of the window stack. If a value of 0 is used with the above command, the designated window is lowered to the bottom of the window stack.

The following program called `raise_w` (found in `/usr/lib/rmb/demo`) uses the keyword `CONTROL` to access register number 22 to move a window to the top of the window stack.

```
100  CREATE WINDOW 601,200,250,640,400;LABEL "One"    ! Makes window 601.
110  CREATE WINDOW 602,220,270,640,400;LABEL "Two"    ! Makes window 602.
120  CREATE WINDOW 603,240,290,640,400;LABEL "Three" ! Makes window 603.
130  !
140  WAIT 2                      ! This allows you time to look at the current
150                                ! window stack.
160  CONTROL 601,22;1 ! Raise window 601 to the top of the window stack.
170  END
```

If you wanted to move window 603 to the bottom of the window stack, you would change line 160 in the above program to read as follows:

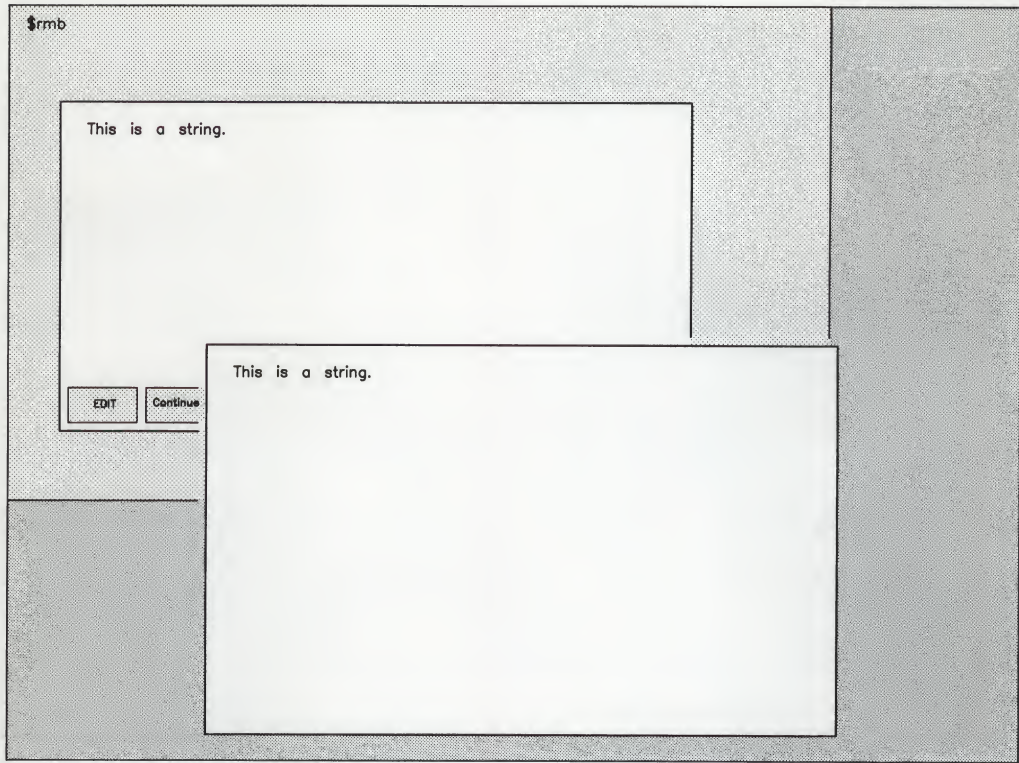
```
160  CONTROL 603,22;0 ! Push window 603 to the bottom of the window stack.
```



## Copying Data Between Windows

The following program `tran_w` (found in `/usr/lib/rmb/demo`) copies the alpha contents of window 601 into the root window. To do this, window 601 is created and assigned as the printing device. Next, the `OUTPUT` statement in line 240 is used to send a string of characters to window 601. Lines 270 and 280 position the cursor in window 601 to the beginning of the string and line 290 reads that string into a string variable. The root window is then assigned as the printing device and the contents of the string variable are printed in window 600.

```
100  INTEGER X_coor,Y_coor,Width,Height
110  X_coor=200 ! Assign the x coordinate position in pixels.
120  Y_coor=350 ! Assign the y coordinate position in pixels.
130  Width=640  ! Assign the width of the window in pixels.
140  Height=400 ! Assign the height of the window in pixels.
150  !
160  ! Create window number 601 and label it as window "One".
170  !
180  CREATE WINDOW 601,X_coor,Y_coor,Width,Height;LABEL "One"
190  !
200  ! Send a string to window 601 and read that string back
210  ! into window 600 (the default window).
220  !
230  PRINTER IS 601          ! Assign window 601 to be the printer.
240  OUTPUT 601 USING "K";"This is a string."      ! Send string to
250                                              ! window 601.
260                                              !
270  CONTROL 601,0;1 ! Position cursor in column one of window 601.
280  CONTROL 601,1;1 ! Position cursor in row one of window 601.
290  ENTER 601 USING "K";String$                  ! Read string from
300                                              ! window 601.
310                                              !
320  PRINTER IS 600          ! Assign window 600 to be the printer.
330  PRINT String$          ! Print the string in window 600.
340  END
```



**Figure 17-4. Copying Data between Windows**



# Table of Contents

---

## Appendix A: HP-HIL Appendix

HP-HIL Command Reference .....	A-2
Identify and Describe (IDD) .....	A-2
Read Register (RRG) .....	A-3
Write Register (WRG) .....	A-3
Report Name (RNM) .....	A-4
Report Status (RST) .....	A-4
Extended Describe (EXD) .....	A-5
Report Security Code (RSC) .....	A-5
Disable Keyswitch Autorepeat (DKA) .....	A-6
Enable Keyswitch Autorepeat (EKA 1,EKA 2) .....	A-6
Prompt 1 thru Prompt 7 (PRM 1 .. PRM 7) .....	A-7
Prompt (PRM) .....	A-7
Acknowledge 1 thru Acknowledge 7 (ACK 1 .. ACK 7) .....	A-8
Acknowledge (ACK) .....	A-8
Device-Dependent Commands (DDC 128 .. 239) .....	A-9
Device ID Byte .....	A-10
Describe Record .....	A-13
Extended Describe Record .....	A-17
Poll Record .....	A-20
Report Security Code Record .....	A-23
Accessible Keycode Definitions .....	A-29





## HP-HIL Appendix

---

This appendix contains information necessary for the development of drivers for HP-HIL devices. The contents of this appendix should be used in conjunction with the chapter in this manual entitled “HP-HIL Devices”. This appendix has been divided into the following section:

- HP-HIL Command Reference
- Device ID Byte
- Describe Record
- Extended Describe Record
- Poll Record
- Report Security Code Record
- Accessible Keycode Definitions

For more information on how HP-HIL devices work, order the *HP-HIL Technical Reference Manual* (HP product number 45918A).

---

## HP-HIL Command Reference

This section provides information for each of the existing HP-HIL commands which are supported by BASIC. The usage of the command is given first, followed by a brief listing of the characteristics of the command.

The characteristics of the commands include:

- how or what the command is used for
  - device identification
  - data input
  - data output
- what commands are not supported by most devices
- a verbal description of the operation of the command

If a device does not support a particular command, it will ignore the command when sent to it.

### Identify and Describe (IDD)

Usage:	The IDD command is used to determine the type of the attached devices, as well as some general characteristics of the device required to understand the data it reports.
Characteristics:	Used for device identification.
Description:	A device responds to the IDD command by first transmitting the device ID byte. The Device ID Byte is used to identify the general class of device and the nationality (in the case of a keyboard other than the HP 98203C). After the ID byte, a series of data bytes, referred to as the Describe Record, is transmitted. This record varies in length and is terminated by a null byte. This means that all bytes of the Describe Record are transmitted with the exception of trailing null bytes and these bytes are ignored. See the "Device ID Byte" and "Describe Record" sections of the "HP-HIL Appendix".

## Read Register (RRG)

- Usage:** Read Register provides the System with an alternate method of collecting data from a device supporting RRG. Note that RRG is not supported by most HP-HIL devices.
- Characteristics:** Used for data input.  
Not supported by most devices.
- Description:** A device indicates support of the Read Register command in the Extended Describe Record, also indicating the specific read registers contained in the device. To perform a register read, the System transmits the address of the register to be read, with the Read Register command. The device, upon receiving the command, transmits the contents of the register.
- HP-HIL register errors are not supported. If an HP-HIL register error does occur, the command that caused the error is ignored.
- Devices which do not support the Read Register command will ignore it (no Register I/O Error is sent).

## Write Register (WRG)

- Usage:** Write Register provides a means of setting the contents of individual registers in devices supporting this advanced feature.
- Characteristics:** Used for data output.  
Not supported by most devices.
- Description:** There are two forms of the Write Register command. Devices indicate support of either of these two forms (or both) in the Extended Describe Record. Both Write Register forms are supported to accommodate devices which support only one or the other form, but they are equivalent capabilities as supported by BASIC.
- HP-HIL register errors are not supported. If an HP-HIL register error does occur, the command that caused the error is ignored.
- Devices which do not support the Write Register command will ignore it (no Register I/O error is sent).



## Report Name (RNM)

- Usage: Report Name is used to request a string of up to 15 characters (8-bit ASCII) which would aid in describing the device to the user.
- Characteristics: Used for device identification.  
Not supported by most devices.
- Description: Characters returned are US ASCII. Devices indicate support of the Report Name command in the Extended Describe Record.
- This record varies in length and is terminated by a null byte. This means that all bytes of the Extended Describe Record are transmitted with the exception of trailing null bytes and these bytes are ignored.

## Report Status (RST)

- Usage: Report Status is used to extract device-specific status information from devices configured on the Link.
- Characteristics: Used for data input.  
Not supported by most devices.
- Description: Devices indicate support of the Report Status command in the Extended Describe Record. This record varies in length and is terminated by a null byte. This means that all bytes of the Extended Describe Record are transmitted with the exception of trailing null bytes and these bytes are ignored. Interpretation of the status bytes will necessarily depend upon the device in question.

## **Extended Describe (EXD)**

Usage:	Extended Describe provides additional information concerning more advanced device features which may not be required for basic operation.
Characteristics:	Used for device identification. Not supported by most devices.
Description:	Support of the Extended Describe command is indicated in the Describe Record. Devices supporting the EXD command respond with a series of data bytes referred to as the Extended Describe Record. This record varies in length and is terminated by a null byte. This means that all bytes of the Extended Describe Record are transmitted with the exception of trailing null bytes and these bytes are ignored. Detailed information on the Extended Describe Record can be found in the section of this appendix entitled "Extended Describe Record".

## **Report Security Code (RSC)**

Usage:	The Report Security Code command is used to extract a unique identifier from a device.
Characteristics:	Used for data input. Not supported by most devices.
Description:	Support of the command is indicated in the Describe Record. The Security Code Record consists of a Header and a variable number of bytes of data terminated by a null byte. This means that all bytes of the Describe Record are transmitted with the exception of trailing null bytes and these bytes are ignored. See the "Report Security Code Record" section of this appendix for further information.

## Disable Keyswitch Autorepeat (DKA)

**Usage:** This command is used to disable the “repeating keys” feature in the addressed device, reducing returned data to one report per keyswitch transition.

**Characteristics:** Not supported by most devices.

**Description:** The default condition of devices supporting DKA and EKA AutoRepeat Commands is Keyswitch AutoRepeat Disabled. More advanced key repeat features may be implemented using device specific commands.

Note that this AutoRepeat is independent of the normal Keyboard AutoRepeat implemented by Series 200 and 300 computers.

## Enable Keyswitch Autorepeat (EKA 1,EKA 2)

**Usage:** These two commands are used to enable the “repeating keys” feature in the addressed device (if the feature is supported).

**Characteristics:** Not supported by most devices.

**Description:** When Keyswitch AutoRepeat is enabled, most keys will repeat at the rate of one report every 40 milliseconds. Following a keyswitch down transition, a delay of 200 ms will occur and the key begins to repeat. Modifier keys (**[Shift]**, **[CTRL]**, **[Extend char]**, etc.) will not repeat, while based on the argument of the Enable Keyswitch AutoRepeat command the Cursor Keys (cursor left, right, up, and down) will repeat at either 20 millisecond or 40 millisecond intervals. Most keys repeat by generating repeated down transitions corresponding to the key position being repeated, although repeating cursor keys on an ITF Keyboard will report a keycode of 02(hexadecimal). Since the BASIC system does not recognize 02 as a valid Keycode, the effect is no cursor key autorepeat for either argument with the ITF Keyboard.

Note that this autorepeat is independent of the normal Keyboard AutoRepeat implemented by Series 200 and 300 computers.

## **Prompt 1 thru Prompt 7 (PRM 1 .. PRM 7)**

Usage:	These commands are used to provide an audible or visual stimulus to the user, perhaps indicating that the System is ready for a particular type of input. Although intended to be directly associated with Acknowledge 1 thru Acknowledge 7 and Button 1 thru Button 7, this association is not a requirement.
Characteristics:	Used for data output. Not supported by most devices.
Description:	The Prompts and Acknowledges supported are indicated in the Describe Record. All unsupported Prompts will be treated the same as other unsupported commands.

## **Prompt (PRM)**

Usage:	Intended as a general-purpose stimulus to the user. Prompt is not intended to be associated with a particular Button as are Prompt 1 thru Prompt 7.
Characteristics:	Used for data output. Not supported by most devices.
Description:	A device indicates support of Prompt in the Describe Record.



## **Acknowledge 1 thru Acknowledge 7 (ACK 1 .. ACK 7)**

Usage:	These commands, similar to the Prompt 1 thru Prompt 7 commands, are intended to provide an audible or visual response to the user, and are generally directly associated with the corresponding Prompt and Button of the same number, although this is not a requirement.
Characteristics:	Used for data output. Not supported by most devices.
Description:	Since there is no explicit "Prompt Off" function provided, this functionality may be part of the Acknowledge definition for a particular device.  The Prompts and Acknowledges supported by the devices are indicated in the Describe Record, and all unsupported Prompts will be treated the same as other unsupported commands.

## **Acknowledge (ACK)**

Usage:	Similar to Prompt, Acknowledge is not associated with any particular Button, but is intended merely as a general purpose audio or visual response to the user.
Characteristics:	Used for data output. Not supported by most devices.
Description:	Since there is no explicit "Prompt Off" function provided, this functionality may be part of the Acknowledge definition for a particular device. Support of Prompt and Acknowledge is indicated in the Describe Record.

## **Device-Dependent Commands (DDC 128 .. 239)**

- Usage: A range of 112 commands has been reserved for use as “device-dependent” commands.
- Characteristics: Not supported by most devices.
- Description: These commands are intended for use by devices with special requirements which the other HP-HIL commands do not really support. Devices should use Read and Write Registers and the Prompts and Acknowledges for special functionality where possible.

---

## Device ID Byte

This section defines the device ID bytes for all types of devices currently defined or anticipated and lists the ID numbers which have currently been allocated. Nationalization for Keyboards is given in the second table.

The Device ID Byte is used to identify the general class of device and the nationality (language) in the case of a Keyboard. Since it is not possible to designate the characteristics of all future devices, the ID Byte should be used to identify only the basic type of device and the nationality (for a Keyboard).

The following table gives device ID Byte definitions for general classes of devices (keyboards, absolute positioners, etc.). For keyboard type devices other than the HP 98203C, note that the ID has a range of 00 to 1F. This allows for the nationalization to be embedded in the ID Byte. The table of nationalized ID definitions gives the lower five bits of the ID Byte. Thus a French ITF keyboard (with an ID range of C0 to DF), would report its ID Byte as DB (C0 + 1B).

**Table A-1. Device ID Byte Definitions**

Device Type	ID Range (hexadecimal)	Assigned Device IDs	HP-HIL Device Name	HP Product Number
Keyboard Group 1	A0 .. FF	C0 .. DF A0 .. BF	ITF Keyboard Integral Keyboard	46020/21A —
Absolute Positioners	80 .. 9F	95 94 93	11×11 Graphics Tablet Size-B Digitizer Size-A Digitizer	45911A 46088A 46087A
Relative Positioners	60 .. 7F	66 66 61 61 60	2-Button Mouse 3-Button Mouse Quadrature Port Control Dials Rotary Control Knob	46060A 46060B 46094A 46085A 46083A
Character Entry	40 .. 5F	5C	Barcode Reader	92916A
Other Devices	20 .. 3F	34 30	ID Module Function Box	46084A 46086A

**Table A-2. Keyboard Nationalized ID Definition**

Lower 5 Bits of Device ID Byte (hexadecimal)	Nationality of Keyboard/Keypad
00	Other <sup>1</sup> <sup>2</sup>
01	reserved
02	Kanji
03	Swiss/French
04	Portuguese <sup>2</sup>
05	Arabic <sup>2</sup>
06	Hebrew <sup>2</sup>
07	Canadian/English
08	Turkish <sup>2</sup>
09	Greek <sup>2</sup>
0A	Thai (Thailand) <sup>2</sup>
0B	Italian
0C	Hangul (Korea) <sup>2</sup>
0D	Dutch
0E	Swedish
0F	German
10	Chinese-PRC (China) <sup>2</sup>
11	Chinese-ROC (Taiwan) <sup>2</sup>
12	Swiss/French II
13	Spanish
14	Swiss/German II
15	Belgian (Flemish)
16	Finnish
17	United Kingdom
18	French/Canadian
19	Swiss/German

<sup>1</sup> See the section "Extended Describe Record" for usage.

<sup>2</sup> Not supported by BASIC, treated as US ASCII.



**Table A-2. Keyboard Nationalized ID Definition (continued)**

<b>Lower 5 Bits of Device ID Byte (hexadecimal)</b>	<b>Nationality of Keyboard/Keypad</b>
1A	Norwegian
1B	French
1C	Danish
1D	Katakana
1E	Latin American/Spanish
1F	United States

---

## Describe Record

The Identify and Describe command is used to determine the type of device(s) attached to the Link and also what their characteristics are.

When a device receives the IDD command, the device will respond by returning a device ID byte followed by the Describe Record. The Record consists of 1 to 10 bytes of information. The first byte of the Describe Record is the Describe Record Header. If the device reports positional information, then 2 bytes will follow containing the resolution of the device. If the device is an absolute positioner, then the maximum count per axis is then reported (for each axis), 2 bytes per axis. The last byte of the Describe Record is the I/O Descriptor Byte.

The Describe Record is shown graphically below:

Device ID
Describe Record Header
Number of counts / cm (m) Low Byte
Number of counts / cm (m) High Byte
Maximum Count X-axis Low Byte
Maximum Count X-axis High Byte
Maximum Count Y-axis Low Byte
Maximum Count Y-axis High Byte
Maximum Count Z-axis Low Byte
Maximum Count Z-axis High Byte
I/O Descriptor Byte

Every device will respond to the IDD command with at least 2 bytes of data, the Device ID Byte, and the Describe Record (1 to 10 bytes). Cursor positioning devices and devices containing buttons, proximity detection, and/or prompt/acknowledge functions will need to report additional information. The Describe Record Header contains some information about the device and provides an indicator of how much additional information is to follow the Header. The description of the Describe Record Header follows:

- Bit 7      Set if the device contains two independent sets of coordinate axes. Consider, for example, a device which interfaces two joysticks to HP-HIL, each with its own independent set of X, Y axes. It is assumed, however, that both sets of coordinate axes share common characteristics as identified in the remainder of the record. Default (clear) indicates a maximum of one set of axes.
- Bit 6      Set if the device is to return absolute positional data (unsigned integers). Default (clear) indicates relative data (2's complement).
- Bit 5      Set if the device returns all positional information at 16-bits/axis. Default (clear) is 8-bits/axis.
- Bit 4      Set if the I/O Descriptor Byte is to follow later in the Describe Record. Default (clear) indicates that the device has no buttons, no proximity detection, and no prompt/acknowledge functionality, with no I/O Descriptor Byte to follow.
- Bit 3      Set if the device supports the Extended Describe command. Default (clear) indicates Extended Describe command is not supported.
- Bit 2      Set if the device supports the Report Security Code command. Default (clear) indicates Report Security Code is not supported.
- Bit 1,0    Bit 1 and bit 0 indicate the coordinate axes the device will report. If non-zero, then following the header will be 16 bits describing the resolution of the device, and in the case of an absolute positioner, 16 bits/axis detailing the extent of each coordinate axis.

Bit 1	Bit 0	Axes Reported
0	0	none
0	1	X
1	0	X and Y
1	1	X, Y, and Z

If the Describe Record Header indicates a non-zero number of axes for which the device will report positional information, then following the Header will be 16 bits describing the resolution of the device in counts per centimeter if the device reports data in a 16-bit format, or in counts per meter if 8-bit format. In the case of an absolute positioner, following the Number of Counts/cm (m) will be 16 bits per axis indicating the maximum extent of each axis for which the device reports data, assuming an origin at the lower left. This is the maximum count per axis the device is capable of reporting, based on a minimum value of 0. Note that these values are reported as 16 bits regardless of whether the device indicates 8-bit or 16-bit data reporting format.

The I/O Descriptor Byte indicates the buttons the device will report keycodes for, whether the device has proximity detection, and what Prompt/Acknowledge functions, if any, are implemented in the device. Note that Prompt and Acknowledge are treated as a set, and no device may indicate support of any particular Prompt or Acknowledge without also supporting its counterpart. If none of the above features are implemented, the I/O Descriptor byte may not be transmitted. The following is the definition of the I/O Descriptor byte:

- Bit 7            Set if the device implements the general purpose Prompt and Acknowledge functions. Default (clear) implies these functions are not implemented.
- Bits 6,5,4      Bits 6, 5, and 4 indicate specific Prompt/Acknowledges (Prompt 1 thru 7 and Acknowledge 1 thru 7) implemented by the device. Default (clear) indicates none.

Bit 6	Bit 5	Bit 4	Prompt/Acks. Implemented
0	0	0	none
0	0	1	1
0	1	0	1 and 2
0	1	1	1, 2, and 3
1	0	0	1 thru 4
1	0	1	1 thru 5
1	1	0	1 thru 6
1	1	1	1 thru 7



Bit 3 Set if the device will report the Proximity In/Out keycodes. Default (clear) indicates no proximity detection.

Bits 2,1,0 Bits 2, 1, and 0 indicate the buttons for which the device will report keycodes.

Bit 2	Bit 1	Bit 0	Buttons Reported
0	0	0	none
0	0	1	1
0	1	0	1 and 2
0	1	1	1, 2, and 3
1	0	0	1 thru 4
1	0	1	1 thru 5
1	1	0	1 thru 6
1	1	1	1 thru 7

---

## Extended Describe Record

Support of the Extended Describe command is indicated in the Describe Record Header. The Extended Describe Record provides additional information concerning more advanced features which may not be required for basic operation.

Devices supporting the Extended Describe command respond with a series of data bytes referred to as the Extended Describe Record. The record length may vary from 1 to 15 bytes (although only 6 bytes are currently defined). The Extended Describe Record has the following format:

Extended Describe Record Header
Maximum Read Register Support
Maximum Write Register Support
Maximum Write Buffer Length Low Byte
Maximum Write Buffer Length High Byte
Localization Code

Devices responding to the Extended Describe command return at least 1 byte of data, the Extended Describe Record Header. Devices supporting Read Register or Write Register or those returning a Localization Code will need to report additional information so that their capabilities may be more fully defined. The Extended Describe Record Header both supplies some of the parameters of the device and provides an indication of how much additional information is to follow. The meanings of the individual bits in the Header are as follows:

- Bit 7      Reserved for future use. Default will be clear.
- Bit 6      Set if the Localization Code is supported. If set, then following the Maximum Write Buffer Length High Byte will be one byte indicating the nationality of the device (keyboard). See the table in the previous section for a listing of the Localization Codes. Default (clear) indicates that the Localization Code is not supported.
- Bit 5      Set if the Report Status command is supported. Default (clear) indicates Report Status not supported.
- Bit 4      Set if the Report Name command is supported. Default (clear) indicates Report Name not supported.
- Bit 3      Reserved for future use. Default will be clear.
- Bit 2      Set if Read Register supported. If set, immediately following the Header is a byte indicating the registers supported for reading in the device. Default will be clear, indicating Read Register not supported.
- Bit 1,0    Bit 1 and bit 0 indicate support of the Write Register command. If bit 1 is set, Write Register Type 2 is supported by the device. If bit 0 is set, Write Register Type 1 is supported. If both bits are set, then the device supports both Type 1 and Type 2. If either bit 1 or bit 0 is set, then following in the Record will be information indicating the registers supported for writing in the device. If bit 1 is set, then an additional 16 bits will be returned indicating the maximum number of data bytes which may be written to the device at a time using Write Register Type 2 without data loss.

If the device indicated support for the Read Register command in the Header, then following the Header is a byte indicating the read registers supported by the device. This byte, the Maximum Read Register Supported byte, indicates the largest read register address supported. Note that it is assumed that all addresses less than this maximum are also supported. Thus a byte of 0Fh indicates that the device contains 16 read registers, addressed as read registers 0 thru 15. HP-HIL protocol allows for devices containing up to 128 read registers, addressed as 0 thru 127.

If Write Register (Type 1 or Type 2) support is indicated, then next is a byte indicating the write registers supported. The Maximum Write Register Supported byte indicates the largest write register address supported in the device. It is assumed that all addresses less than the maximum are also supported. Up to 128 write registers, addressed as 0 thru 127, are supported in the HP-HIL protocol.

If Write Register Type 2 is supported, as indicated by bit 1 of the Extended Describe Record Header being set, then following the Maximum Write Register Supported byte is 16 bits of data indicating the maximum number of bytes which may be transmitted to the device in a Type 2 transfer without overflowing the device's internal buffer. This number, transmitted first low byte, then high byte, represents the buffer length of the device minus 1. Thus a device capable of buffering 1024 bytes of data would transmit a Maximum Buffer Length Low Byte of FFh and a Maximum Buffer Length High Byte of 03h.

If the Localization code is supported, then the Localization Code byte will be included in the Extended Describe Record. The Localization Code is an 8 bit number which corresponds to a nationality (language) of a keyboard. The table in the previous section, lists currently assigned Localization Codes and languages (values from 20 through FF are reserved).



---

## Poll Record

The Poll command is the fundamental means for extracting data from the input devices attached to the Link. Data is sent back to the host in the form of a record, which may contain character data, position data, or some status information.

Data returned from HP-HIL devices is in record form, similar to the response to the Describe command. Each device transmits its individual Poll Record. Note that it may not be required for the device to report all available information in response to a single Poll request; data may be split between Polls provided correct formatting is observed for each record reported. The Poll Record is structured as follows:

Poll Record Header
X-axis Data Low Byte
X-axis Data High Byte
Y-axis Data Low Byte
Y-axis Data High Byte
Z-axis Data Low Byte
Z-axis Data High Byte
Character Data
:
Character Data

The function of the Poll Record Header is to indicate to the System the type and quantity of information to follow, as well as to report simple status information. The bits of the Header are assigned as follows:

- Bit 7            Set if the device is reporting data from the second set of coordinate axes. Default (clear) indicates data from set 1.
- Bit 6,5,4       Based on the value of these 3 bits, following all position information will be character data (up to 8 bytes):

Bit 6	Bit 5	Bit 4	Character Data Description
0	0	0	No character data to follow
0	0	1	Reserved Character Set 1
0	1	0	US ASCII Characters
0	1	1	Binary Data
1	0	0	Keycode Set 1
1	0	1	Reserved Character Set 2
1	1	0	Keycode Set 2 *
1	1	1	Keycode Set 3

\* These keycodes are device dependent.  
They use the LSB to indicate the key transition (0 = Down, 1 = Up). 126 keys maximum.

- Bit 3            Set indicates request for status check. Clear (default) indicates status unchanged.
- Bit 2            Set indicates device ready for data. Default (clear) indicates not ready for data transfer at this time.
- Bit 1,0          Bit 1 and bit 0 indicate the coordinate axes the device is reporting:

Bit 1	Bit 0	Axes Reported
0	0	none
0	1	X
1	0	X and Y
1	1	X, Y, and Z

Following the Header is the device data. If the device indicated that it would report coordinate information 16-bits/axis in the Describe Record, then for each axis reported will be first the low, then high byte coordinate data. Otherwise, the high byte will not be transmitted. In general, the Poll Record format indicates the maximum data which can be reported; most devices will transmit only a subset each time. Following the positional information will be up to 8 bytes of character data, as specified in the Poll Record Header. The different types of character data may not be mixed. Note that more than one device may respond to the Poll command; each will respond with an individual Poll Record, distinguishable from the previous by the address field of the data.

The BASIC system automatically sends poll commands at approximately 20 millisecond intervals.

---

## Report Security Code Record

The Report Security Code command is used to extract a unique identifier from the device. Support of the command is indicated in the Describe Record Header. The Report Security Code Record consists of a header (1 byte) which defines the format of the data following the header (the remaining 1 to 14 bytes of data). Bits 7 through 4 of the header byte describe the data format type. Currently, only one data format type is defined, Type 1. Bits 3 through 0 are reserved, and should be set to 0. Thus the only currently valid header is for a Type 1 format (hexadecimal 10). The Report Security Code Record is similar in purpose to a serial number, it may also contain information related to user identity, network address, or other information which is unique to a particular user or environment.

The only data transmitted by the ID Module is in response to the Report Security Command. However, the following information applies to any device that supports the Report Security command.

The data format consists of a one byte header and eight bytes of binary data. The eight data bytes are the packed product and serial numbers of the HP-HIL device. In the case where an ID Module is an exchange module signified by a ten digit part number, the five digit prefix number remains the same and the product number letter is replaced by the least significant digit of the part number.



The product, exchange and serial number formats are:

Header	:	H	(1 byte header)
Product number is	:	DDDDDA	(5 digits and 1 ASCII character)
Exchange part number	:	DDDDd	(5 digits and 1 ASCII character)
Serial number is	:	YYWW©NNNNN	(9 digits and 1 ASCII character)

where:

H	is the data header.
DDDDD	is the product number (e.g. 46084).
A	is the product number alpha character.
d	is the least significant numeric character of the exchange number.
YY	is the year code (year less 60).
WW	is the week code (0 to 51).
©	is the serial country of manufacturing code.
NNNNN	is the serial suffix (0 to 99 999)

The header byte is transmitted before the eight data bytes. The header's purpose is to allow for other data formats, however none are currently implemented.

The five digits of the product or exchange part prefix number are converted to a two byte binary number and the high order bit of a third byte. The remaining lower seven bits of the third byte contain the ASCII character. In products where two alpha characters are used in the product number, only the first character is used in the data format. The order of the bytes have been arranged to transmit the least significant byte of the number first.

In a similar manner, the nine digits of the serial number are converted to a four byte binary number. The country code of manufacturing is in the last byte to be transmitted and is an ASCII character.

The Report Security data bytes are transmitted in the following order, starting with byte 1 and going through byte 9. Bits are numbered starting with bit 0 at the right most position of the byte (least significant bit) and going through bit 7 (most significant bit), left most position.

Header (10 hexadecimal)	
Product Number Bits 7 .. 0	
Product Number Bits 15 .. 8	
Product Number Bit 16	Product Letter Suffix ASCII (7 Bits)
Serial Number Bits 7 .. 0	
Serial Number Bits 15 .. 8	
Serial Number Bits 23 .. 16	
0 0	Serial Number Bits 29 .. 24
0	Country of Manufacture USASCII (7 Bits)

The report security data format is:

Byte	Bit(s)	Description
1	7 - 0	The first byte is the header containing the number 10 hexadecimal for the following format. The general scheme for the header is:  Bits 7 - 4 are assigned as format variations, where format 1 is the only assignment.  Bits 3 - 0 are undefined, but set to zero.
2	7 - 0	The second and third bytes and the 7th bit of the fourth byte represent the 5 digits of the product or exchange part number DDDDD in binary form. The least significant bit is bit 0 of byte two.
3	7 - 0	
4	7	
4	6 - 0	The least significant seven bits of byte four represent the product letter or the least significant digit of the exchange number numeric character. The character is the US ASCII 7 bit representation of the character.
5	7 - 0	The fifth, sixth, seventh bytes and the six least significant bits of byte eight represent the 9 digits of the serial number YYWWNNNNN in binary form, without the alpha character. The least significant bit is bit 0 of byte 5.
6	7 - 0	
7	7 - 0	
8	5 - 0	
8	7 - 6	The two most significant bits of byte eight are reserved for future use and are set to zero.
9	6 - 0	The least significant seven bits of byte 9 represent the serial number letter. The character is the US ASCII 7 bit representation of the character.
9	7	The most significant bit of byte nine is reserved for future use and is set to zero.

### Sample of Report Security Format for A Product Module

The following information is returned upon receiving a Report Security command for a Product Module. The data is based on the data format described in the last section. Byte 1 is the first byte sent from the module to the host.

The sample results given are based on the product number 46084A and serial number 2519A00001. The serial number corresponds with the year of 1985, week 19, and serial number suffix 00001. Note that by adding 60 to the above serial numbers first two digits you get the year 85.

Byte No.	Data (hex)	Description
1	10	Header
2	04	Part of product number 46084
3	B4	Part of product number 46084
4	41	Product letter "A" and part of product number 46084
5	61	Part of serial number
6	B0	Part of serial number
7	03	Part of serial number
8	0F	Part of serial number
9	41	Country of Manufacturing Code



### Sample of Report Security Format for An Exchange Module

The following information is returned upon receiving a Report Security command for an Exchange Module. The data is based on the data format described in the section, "Report Security Code (RSC)". Byte 1 is the first byte sent from the module to the host.

The sample results given are based on the exchange number 46084-69901 and serial number 2519A00001. The serial number corresponds with the year of 1985, week 19, and serial number suffix 00001. Note that by adding 60 to the above serial numbers first two digits you get the year 85.

Byte No.	Data (hex)	Description
1	10	Header
2	04	Part of product number 46084
3	B4	Part of product number 46084
4	31	US ASCII character "1" which is part of the product number 460841
5	61	Part of serial number
6	B0	Part of serial number
7	03	Part of serial number
8	0F	Part of serial number
9	41	Country of Manufacturing Code

Since the sample is an exchange module, the exchange part number transmitted is 460841. Byte 4 is the hexadecimal value of 31 which represent the US ASCII character "1". Note, the prefix number 46084 does not change from the sample of the product module and the character "1" is really an ASCII character.

---

## Accessible Keycode Definitions

This section covers a subset of Keycode Set 1. Keycode Set 1 provides the keycodes for the down and up keystrokes of ITF keyboards (HP 46020/21A). Note that both the ITF Keycode Set 1 and the version of Keycode Set 2 used by the HP 98203C keyboard are always processed by the system and thus are never available through the buffer used by the HILBUF\$ function. The subset of Keycode Set 1 which is used by HP-HIL Graphics Tablets is the small portion of Keycode Set 1 covered in this section. The HP 46066A Function Box uses Keycode Set 2. Its keys are numbered 0 through 31 starting with the upper-left key and going from left to right, then down. Key down generates Keycode  $2 \times n$  and key up generates Keycode  $2 \times n + 1$  where  $n$  is the key number.

**Table A-3. Keycode Set 1**

Keycode for Down Transition (hex)	Keycode for Up Transition (hex)	United States Keycap Legend	Notes
80	81	<BUTTON 1>	1
82	83	<BUTTON 2>	1
84	85	<BUTTON 3>	1
86	87	<BUTTON 4>	1
88	89	<BUTTON 5>	1
8A	8B	<BUTTON 6>	1
8C	8D	<BUTTON 7>	1
8E	8F	<PROXIMITY IN/OUT>	1

---

<sup>1</sup> Typically used in positioning devices and not found on keyboards.



# Table of Contents

---

## Appendix B: Useful Tables

Option Numbers .....	B-1
Interface Select Codes .....	B-2
Display-Enhancement Characters .....	B-3
Monochrome Enhancements .....	B-3
Color Enhancements .....	B-4
U.S. ASCII Character Codes .....	B-5
U.S. ASCII Character Codes .....	B-6
U.S./European Display Characters .....	B-7
U.S./European Display Characters .....	B-8
U.S./European Display Characters .....	B-9
U.S./European Display Characters .....	B-10
U.S./European Display Characters .....	B-11
U.S./European Display Characters .....	B-12
Katakana Display Characters .....	B-13
Katakana Display Characters .....	B-14
Katakana Display Characters .....	B-15
Katakana Display Characters .....	B-16
Master Reset Table .....	B-17
Graphic Reset Table .....	B-20
Interface Reset Table .....	B-21
Selected High-Precision Metric Conversion Factors .....	B-23



# Memorandum for Mr. [Name]

Subject: [Topic]

Reference: [Reference]

Date: [Date]

By: [Name]

For: [Name]

File: [File]

Page: [Page]

Version: [Version]

Status: [Status]

Comments: [Comments]

Attachments: [Attachments]

History: [History]

Notes: [Notes]

Summary: [Summary]

Conclusion: [Conclusion]

Recommendation: [Recommendation]

Next Steps: [Next Steps]

Follow-up: [Follow-up]

Review: [Review]

Approval: [Approval]

Signature: [Signature]

Date: [Date]

Location: [Location]

Time: [Time]

Weather: [Weather]

Temperature: [Temperature]

Humidity: [Humidity]

Wind: [Wind]

Clouds: [Clouds]

Visibility: [Visibility]

Pressure: [Pressure]

Altitude: [Altitude]

Latitude: [Latitude]

Longitude: [Longitude]

Time Zone: [Time Zone]

Daylight Saving Time: [Daylight Saving Time]

## Useful Tables

### Option Numbers

These option numbers are displayed when ERROR 1 is reported.

Option Number	Binary	Option Number	Binary
1	BASIC Main	21	CS80
2	GRAPH	22	BUBBLE
3	GRAPHX	23	EPROM
4	IO	24	HP 9885
5	BASIC Main	25	HPIB
6	TRANS	26	FHPIB
7	MAT	27	SERIAL
8	PDEV	28	GPIO
9	XREF	29	BCD
10	KBD	30	DCOMM
11	CLOCK	31-40	Reserved
12	LEX	41	"Unavailable"
13	BASIC Main	42	CRTB
14	MS	43	CRTA
15	SRM	44-45	Reserved
16	Reserved	46	COMPLEX
17	PCIB <sup>1</sup>	47	CRTX
18	KNB2_0	48	EDIT
19	ERR	49	Reserved
20	DISC	50	HFS

<sup>1</sup> This binary is included in the support software for the HP 98647 PC Instruments Interface. It is not supplied with the BASIC 5.0 system.

---

## Interface Select Codes

### Internal Select Codes

Select Code	Device or Interface
1	Display (alpha)
2	Keyboard
3	Display (graphics)
4	Internal floppy-disc drive
5	Optional powerfail protection interface
6	Display (Graphics for bit mapped)
7	HP-IB interface (built-in)

### Factory Presets for External Interfaces

Select Code	Device or Interface
8	HP-IB
9	RS-232
10	(not used)
11	BCD
12	GPIO
14	HP-IB "High-Speed" Disc Interface
20	Data Communications
21	Shared Resource Management
27	EPROM Programmer
28	RGB Color Video
30	Bubble Memory
32	Parity, Cache, Floating-point math hardware, and battery-backed clock (Pseudo Select Code)

---

## Display-Enhancement Characters

Displaying these characters on the CRT (with OUTPUT CRT, PRINT, or DISP, etc.) produce special effects.

### Monochrome Enhancements

These characters produce special effects on most monochrome displays.

Character Code	Action Resulting from Displaying the Character
128	All enhancements off.
129	Inverse mode on.
130	Blinking mode on. *
131	Inverse and Blinking modes on. *
132	Underline mode on.
133	Underline and Inverse modes on.
134	Underline and Blinking modes on. *
135	Underline, Inverse, and Blinking modes on. *

\* *Blinking not available on bit-mapped alpha displays.*



## Color Enhancements

These characters change the alpha pen color on color displays.

Character Code	Model 236C Display	Bit-mapped Alpha Display
136	White	PEN 1
137	Red	PEN 2
138	Yellow	PEN 3
139	Green	PEN 4
140	Cyan	PEN 5
141	Blue	PEN 6
142	Magenta	PEN 7
143	Black	PEN 0

CRT CONTROL registers 5 and 15 through 17 also provide a method of changing the alpha color.

PRINTing  $\text{CHR}\$(x)$ , where  $136 \leq x \leq 143$ , will provide the same colors as on the Model 236C as long as the color map contains default values and the alpha write-enable mask includes planes 0 through 2. A user-defined color map which changes the values of pens 0 to 7 will change the meaning of  $\text{CHR}\$(x)$ .

# U.S. ASCII Character Codes

STD-LL-50182

ASCII Char.	EQUIVALENT FORMS				HP-IB
	Dec	Binary	Oct	Hex	
NUL	0	00000000	000	00	
SOH	1	00000001	001	01	GTL
STX	2	00000010	002	02	
ETX	3	00000011	003	03	
EOT	4	00000100	004	04	SDC
ENQ	5	00000101	005	05	PPC
ACK	6	00000110	006	06	
BEL	7	00000111	007	07	
BS	8	00001000	010	08	GET
HT	9	00001001	011	09	TCT
LF	10	00001010	012	0A	
VT	11	00001011	013	0B	
FF	12	00001100	014	0C	
CR	13	00001101	015	0D	
SO	14	00001110	016	0E	
SI	15	00001111	017	0F	
DLE	16	00010000	020	10	
DC1	17	00010001	021	11	LLO
DC2	18	00010010	022	12	
DC3	19	00010011	023	13	
DC4	20	00010100	024	14	DCL
NAK	21	00010101	025	15	PPU
SYNC	22	00010110	026	16	
ETB	23	00010111	027	17	
CAN	24	00011000	030	18	SPE
EM	25	00011001	031	19	SPD
SUB	26	00011010	032	1A	
ESC	27	00011011	033	1B	
FS	28	00011100	034	1C	
GS	29	00011101	035	1D	
RS	30	00011110	036	1E	
US	31	00011111	037	1F	

ASCII Char.	EQUIVALENT FORMS				HP-IB
	Dec	Binary	Oct	Hex	
space	32	00100000	040	20	LA0
!	33	00100001	041	21	LA1
"	34	00100010	042	22	LA2
#	35	00100011	043	23	LA3
\$	36	00100100	044	24	LA4
%	37	00100101	045	25	LA5
&	38	00100110	046	26	LA6
'	39	00100111	047	27	LA7
(	40	00101000	050	28	LA8
)	41	00101001	051	29	LA9
*	42	00101010	052	2A	LA10
+	43	00101011	053	2B	LA11
,	44	00101100	054	2C	LA12
-	45	00101101	055	2D	LA13
.	46	00101110	056	2E	LA14
/	47	00101111	057	2F	LA15
0	48	00110000	060	30	LA16
1	49	00110001	061	31	LA17
2	50	00110010	062	32	LA18
3	51	00110011	063	33	LA19
4	52	00110100	064	34	LA20
5	53	00110101	065	35	LA21
6	54	00110110	066	36	LA22
7	55	00110111	067	37	LA23
8	56	00111000	070	38	LA24
9	57	00111001	071	39	LA25
:	58	00111010	072	3A	LA26
;	59	00111011	073	3B	LA27
<	60	00111100	074	3C	LA28
=	61	00111101	075	3D	LA29
>	62	00111110	076	3E	LA30
?	63	00111111	077	3F	UNL

# U.S. ASCII Character Codes

ASCII Char.	EQUIVALENT FORMS				HP-IB
	Dec	Binary	Oct	Hex	
@	64	01000000	100	40	TA0
A	65	01000001	101	41	TA1
B	66	01000010	102	42	TA2
C	67	01000011	103	43	TA3
D	68	01000100	104	44	TA4
E	69	01000101	105	45	TA5
F	70	01000110	106	46	TA6
G	71	01000111	107	47	TA7
H	72	01001000	110	48	TA8
I	73	01001001	111	49	TA9
J	74	01001010	112	4A	TA10
K	75	01001011	113	4B	TA11
L	76	01001100	114	4C	TA12
M	77	01001101	115	4D	TA13
N	78	01001110	116	4E	TA14
O	79	01001111	117	4F	TA15
P	80	01010000	120	50	TA16
Q	81	01010001	121	51	TA17
R	82	01010010	122	52	TA18
S	83	01010011	123	53	TA19
T	84	01010100	124	54	TA20
U	85	01010101	125	55	TA21
V	86	01010110	126	56	TA22
W	87	01010111	127	57	TA23
X	88	01011000	130	58	TA24
Y	89	01011001	131	59	TA25
Z	90	01011010	132	5A	TA26
[	91	01011011	133	5B	TA27
\	92	01011100	134	5C	TA28
]	93	01011101	135	5D	TA29
^	94	01011110	136	5E	TA30
_	95	01011111	137	5F	UNT

ASCII Char.	EQUIVALENT FORMS				HP-IB
	Dec	Binary	Oct	Hex	
`	96	01100000	140	60	SC0
a	97	01100001	141	61	SC1
b	98	01100010	142	62	SC2
c	99	01100011	143	63	SC3
d	100	01100100	144	64	SC4
e	101	01100101	145	65	SC5
f	102	01100110	146	66	SC6
g	103	01100111	147	67	SC7
h	104	01101000	150	68	SC8
i	105	01101001	151	69	SC9
j	106	01101010	152	6A	SC10
k	107	01101011	153	6B	SC11
l	108	01101100	154	6C	SC12
m	109	01101101	155	6D	SC13
n	110	01101110	156	6E	SC14
o	111	01101111	157	6F	SC15
p	112	01110000	160	70	SC16
q	113	01110001	161	71	SC17
r	114	01110010	162	72	SC18
s	115	01110011	163	73	SC19
t	116	01110100	164	74	SC20
u	117	01110101	165	75	SC21
v	118	01110110	166	76	SC22
w	119	01110111	167	77	SC23
x	120	01111000	170	78	SC24
y	121	01111001	171	79	SC25
z	122	01111010	172	7A	SC26
{	123	01111011	173	7B	SC27
	124	01111100	174	7C	SC28
}	125	01111101	175	7D	SC29
~	126	01111110	176	7E	SC30
DEL	127	01111111	177	7F	SC31

# U.S./European Display Characters

These characters can be displayed on the alpha screens of Models 216, 220 (with a 98204A display), 226, and 236 Computers.

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
N	0	00000000
h	1	00000001
S	2	00000010
E	3	00000011
F	4	00000100
Q	5	00000101
R	6	00000110
U	7	00000111
B	8	00001000
M	9	00001001
L	10	00001010
V	11	00001011
W	12	00001100
K	13	00001101
O	14	00001110
X	15	00001111
P	16	00010000
A	17	00010001
D	18	00010010
G	19	00010011
J	20	00010100
N	21	00010101
S	22	00010110
F	23	00010111
T	24	00011000
H	25	00011001
B	26	00011010
F	27	00011011
V	28	00011100
Q	29	00011101
U	30	00011110
W	31	00011111
!	32	00100000
"	33	00100001
"	34	00100010
#	35	00100011
\$	36	00100100
%	37	00100101
&	38	00100110
'	39	00100111
<	40	00101000
>	41	00101001
*	42	00101010
+	43	00101011
,	44	00101100
-	45	00101101
.	46	00101110
/	47	00101111
0	48	00110000
1	49	00110001
2	50	00110010
3	51	00110011
4	52	00110100
5	53	00110101
6	54	00110110
7	55	00110111
8	56	00111000
9	57	00111001
:	58	00111010
;	59	00111011
<	60	00111100
=	61	00111101
>	62	00111110
?	63	00111111
@	64	01000000
A	65	01000001
B	66	01000010
C	67	01000011
D	68	01000100
E	69	01000101
F	70	01000110
G	71	01000111
H	72	01001000
I	73	01001001
J	74	01001010
K	75	01001011
L	76	01001100
M	77	01001101
N	78	01001110
O	79	01001111
P	80	01010000
Q	81	01010001
R	82	01010010
S	83	01010011
T	84	01010100
U	85	01010101
V	86	01010110
W	87	01010111
X	88	01011000
Y	89	01011001
Z	90	01011010
[	91	01011011
\	92	01011100
]	93	01011101
^	94	01011110
_	95	01011111
`	96	01100000
a	97	01100001
b	98	01100010
c	99	01100011
d	100	01100100
e	101	01100101
f	102	01100110
g	103	01100111
h	104	01101000
i	105	01101001
j	106	01101010
k	107	01101011
l	108	01101100
m	109	01101101
n	110	01101110
o	111	01101111
p	112	01110000
q	113	01110001
r	114	01110010
s	115	01110011
t	116	01110100
u	117	01110101
v	118	01110110
w	119	01110111
x	120	01111000
y	121	01111001
z	122	01111010
{	123	01111011
	124	01111100
}	125	01111101
~	126	01111110
~	127	01111111



# U.S./European Display Characters

These characters can be displayed on the alpha screens of Models 216, 220 (with a 98204A display), 226, and 236 Computers.

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
h	128	10000000
h	129	10000001
h	130	10000010
h	131	10000011
h	132	10000100
h	133	10000101
h	134	10000110
h	135	10000111
h	136	10001000
h	137	10001001
h	138	10001010
h	139	10001011
h	140	10001100
h	141	10001101
h	142	10001110
h	143	10001111
h	144	10010000
h	145	10010001
h	146	10010010
h	147	10010011
h	148	10010100
h	149	10010101
h	150	10010110
h	151	10010111
h	152	10011000
h	153	10011001
h	154	10011010
h	155	10011011
h	156	10011100
h	157	10011101
h	158	10011110
h	159	10011111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
h	160	10100000
A	161	10100001
A	162	10100010
E	163	10100011
E	164	10100100
E	165	10100101
I	166	10100110
I	167	10100111
^	168	10101000
^	169	10101001
^	170	10101010
^	171	10101011
^	172	10101100
U	173	10101101
U	174	10101110
E	175	10101111
—	176	10110000
Y	177	10110001
Y	178	10110010
o	179	10110011
C	180	10110100
ç	181	10110101
N	182	10110110
N	183	10110111
i	184	10111000
ç	185	10111001
Q	186	10111010
E	187	10111011
h	188	10111100
S	189	10111101
h	190	10111110
h	191	10111111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
ä	192	11000000
ë	193	11000001
ö	194	11000010
ü	195	11000011
ä	196	11000100
ë	197	11000101
ö	198	11000110
ü	199	11000111
ä	200	11001000
ë	201	11001001
ö	202	11001010
ü	203	11001011
ä	204	11001100
ë	205	11001101
ö	206	11001110
ü	207	11001111
Ä	208	11010000
Ë	209	11010001
Ö	210	11010010
Ü	211	11010011
ä	212	11010100
í	213	11010101
ö	214	11010110
æ	215	11010111
Ä	216	11011000
Ë	217	11011001
Ö	218	11011010
Ü	219	11011011
É	220	11011100
Ï	221	11011101
ß	222	11011110
Ü	223	11011111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
Ä	224	11100000
Ä	225	11100001
ä	226	11100010
D	227	11100011
d	228	11100100
I	229	11100101
I	230	11100110
Ü	231	11100111
Ü	232	11101000
Ü	233	11101001
Ü	234	11101010
S	235	11101011
S	236	11101100
U	237	11101101
Y	238	11101110
Y	239	11101111
Y	240	11110000
Y	241	11110001
h	242	11110010
h	243	11110011
h	244	11110100
h	245	11110101
h	246	11110110
h	247	11110111
h	248	11111000
h	249	11111001
h	250	11111010
h	251	11111011
h	252	11111100
h	253	11111101
h	254	11111110
h	255	11111111

Note 1: Characters 128 through 135 are "monochrome highlights" (see preceding table).

Note 2: Characters 136 through 143 are "color highlights" (see preceding table).

Note 3: Characters 144 through 159 are ignored by PRINT and DISP statements.

## U.S./European Display Characters

These characters can be displayed on the alpha screen of Series 200 Model 217, 220 (with 98204B display), and 237 computers, and on Series 300 computers using a 98546 Display Compatibility Interface or 98700 Display Controller.

### ASCII

Num.	Chr.	Num.	Chr.	Num.	Chr.	Num.	Chr.
0	N	32		64	@	96	`
1	H	33	!	65	A	97	a
2	X	34	"	66	B	98	b
3	X	35	#	67	C	99	c
4	T	36	\$	68	D	100	d
5	Q	37	%	69	E	101	e
6	K	38	&	70	F	102	f
7	A	39	'	71	G	103	g
8	S	40	(	72	H	104	h
9	H	41	)	73	I	105	i
10	F	42	*	74	J	106	j
11	V	43	+	75	K	107	k
12	F	44	,	76	L	108	l
13	R	45	-	77	M	109	m
14	O	46	.	78	N	110	n
15	I	47	/	79	O	111	o
16	L	48	0	80	P	112	p
17	D <sub>1</sub>	49	1	81	Q	113	q
18	D <sub>2</sub>	50	2	82	R	114	r
19	D <sub>3</sub>	51	3	83	S	115	s
20	D <sub>4</sub>	52	4	84	T	116	t
21	NK	53	5	85	U	117	u
22	Y	54	6	86	V	118	v
23	B	55	7	87	W	119	w
24	CN	56	8	88	X	120	x
25	M	57	9	89	Y	121	y
26	B	58	:	90	Z	122	z
27	C	59	;	91	[	123	{
28	F	60	<	92	\	124	
29	S	61	=	93	]	125	}
30	U	62	>	94	^	126	~
31	U	63	?	95	_	127	■

## U.S./European Display Characters

These characters can be displayed on the alpha screen of Series 200 Model 217, 220 (with 98204B display), and 237 computers, and on Series 300 computers using a 98546 Display Compatibility Interface or 98700 Display Controller.

### ASCII I

Num.	Chr.	Num.	Chr.	Num.	Chr.	Num.	Chr.
128	C	160		192	à	224	À
129	L	161	À	193	é	225	Á
130	I	162	Á	194	ò	226	Â
131	U	163	Ê	195	û	227	Ë
132	B	164	Ë	196	á	228	Ď
133	G	165	Ê	197	é	229	í
134	T	166	±	198	ó	230	ì
135	V	167	±	199	ú	231	ó
136	L	168	'	200	à	232	ò
137	H	169	,	201	è	233	Ë
138	R	170	^	202	ò	234	Ö
139	D	171	"	203	ù	235	Š
140	Y	172	~	204	ä	236	š
141	E	173	Ù	205	è	237	Ú
142	R	174	Ù	206	ö	238	Ÿ
143	G	175	£	207	ü	239	ÿ
144	K	176	—	208	À	240	Þ
145	S	177	B <sub>1</sub>	209	í	241	þ
146	O	178	B <sub>2</sub>	210	Ø	242	F <sub>2</sub>
147	S <sub>1</sub>	179	.	211	Æ	243	F <sub>3</sub>
148	S <sub>2</sub>	180	Ç	212	à	244	F <sub>4</sub>
149	S <sub>3</sub>	181	Ç	213	í	245	I <sub>0</sub>
150	S <sub>4</sub>	182	Ñ	214	ø	246	—
151	S <sub>5</sub>	183	Ñ	215	æ	247	¼
152	S <sub>6</sub>	184	í	216	À	248	½
153	S <sub>7</sub>	185	¿	217	l	249	¾
154	S <sub>8</sub>	186	Ð	218	Ö	250	Ω
155	S <sub>9</sub>	187	£	219	Ü	251	«
156	S <sub>A</sub>	188	¥	220	É	252	■
157	S <sub>B</sub>	189	§	221	İ	253	»
158	S <sub>C</sub>	190	f	222	ß	254	±
159	S <sub>D</sub>	191	ç	223	Ö	255	☒



## U.S./European Display Characters

These characters can be displayed on the screen of Series 300 computers (except with a 98546 Display Compatibility Interface or 98700 Display Controller; see the preceding table).

### ASCII

Num.	Chr.	Num.	Chr.	Num.	Chr.	Num.	Chr.
0	N	32		64	@	96	`
1	H	33	!	65	A	97	a
2	X	34	"	66	B	98	b
3	E	35	#	67	C	99	c
4	T	36	\$	68	D	100	d
5	Q	37	%	69	E	101	e
6	K	38	&	70	F	102	f
7	U	39	'	71	G	103	g
8	S	40	(	72	H	104	h
9	I	41	)	73	I	105	i
10	F	42	*	74	J	106	j
11	V	43	+	75	K	107	k
12	P	44	,	76	L	108	l
13	R	45	-	77	M	109	m
14	O	46	.	78	N	110	n
15	1	47	/	79	O	111	o
16	L	48	0	80	P	112	p
17	0	49	1	81	Q	113	q
18	2	50	2	82	R	114	r
19	3	51	3	83	S	115	s
20	4	52	4	84	T	116	t
21	K	53	5	85	U	117	u
22	Y	54	6	86	V	118	v
23	B	55	7	87	W	119	w
24	N	56	8	88	X	120	x
25	M	57	9	89	Y	121	y
26	B	58	:	90	Z	122	z
27	C	59	;	91	[	123	{
28	F	60	<	92	\	124	
29	S	61	=	93	]	125	}
30	U	62	>	94	^	126	~
31	U	63	?	95	_	127	■



## U.S./European Display Characters

These characters can be displayed on the screen of Series 300 computers (except with a 98546 Display Compatibility Interface or 98700 Display Controller; see the preceding table).

### ASCII

Num.	Chr.	Num.	Chr.	Num.	Chr.	Num.	Chr.
128	CL	160		192	à	224	Á
129	IV	161	À	193	ê	225	Â
130	BG	162	Â	194	ô	226	Ã
131	IB	163	Ê	195	û	227	Ä
132	UL	164	É	196	á	228	Å
133	IV	165	È	197	é	229	Í
134	BG	166	Ë	198	ó	230	Ì
135	IV	167	Ï	199	ú	231	Ó
136	WH	168	´	200	à	232	Ò
137	RD	169	`	201	è	233	Ö
138	YE	170	^	202	ò	234	Ø
139	GR	171	¨	203	ù	235	Š
140	CY	172	~	204	ä	236	š
141	BU	173	Ù	205	ë	237	Ú
142	MG	174	Ö	206	ö	238	Û
143	BK	175	£	207	ü	239	Ü
144	SO	176	—	208	À	240	Þ
145	91	177	Ý	209	Í	241	þ
146	92	178	Ý	210	Ø	242	·
147	93	179	·	211	À	243	µ
148	94	180	Ç	212	à	244	¶
149	95	181	Ç	213	í	245	¯
150	96	182	Ñ	214	ø	246	—
151	97	183	ñ	215	æ	247	¼
152	98	184	ì	216	Ä	248	½
153	99	185	í	217	Ì	249	¾
154	9A	186	Ò	218	Ö	250	º
155	9B	187	£	219	Ü	251	«
156	9C	188	¥	220	É	252	■
157	9D	189	§	221	Ï	253	»
158	9E	190	ƒ	222	ß	254	±
159	9F	191	¢	223	Ô	255	⊞

# Katakana Display Characters

These characters can be displayed on the screen of Model 216, 217, 220, 226, and 236 computers, and on Series 300 computers using a 98546 Display Compatibility Interface.

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
0	0	00000000
1	1	00000001
2	2	00000010
3	3	00000011
4	4	00000100
5	5	00000101
6	6	00000110
7	7	00000111
8	8	00001000
9	9	00001001
10	10	00001010
11	11	00001011
12	12	00001100
13	13	00001101
14	14	00001110
15	15	00001111
16	16	00010000
17	17	00010001
18	18	00010010
19	19	00010011
20	20	00010100
21	21	00010101
22	22	00010110
23	23	00010111
24	24	00011000
25	25	00011001
26	26	00011010
27	27	00011011
28	28	00011100
29	29	00011101
30	30	00011110
31	31	00011111
32	32	00100000
33	33	00100001
34	34	00100010
35	35	00100011
36	36	00100100
37	37	00100101
38	38	00100110
39	39	00100111
40	40	00101000
41	41	00101001
42	42	00101010
43	43	00101011
44	44	00101100
45	45	00101101
46	46	00101110
47	47	00101111
48	48	00110000
49	49	00110001
50	50	00110010
51	51	00110011
52	52	00110100
53	53	00110101
54	54	00110110
55	55	00110111
56	56	00111000
57	57	00111001
58	58	00111010
59	59	00111011
60	60	00111100
61	61	00111101
62	62	00111110
63	63	00111111
64	64	01000000
65	65	01000001
66	66	01000010
67	67	01000011
68	68	01000100
69	69	01000101
70	70	01000110
71	71	01000111
72	72	01001000
73	73	01001001
74	74	01001010
75	75	01001011
76	76	01001100
77	77	01001101
78	78	01001110
79	79	01001111
80	80	01010000
81	81	01010001
82	82	01010010
83	83	01010011
84	84	01010100
85	85	01010101
86	86	01010110
87	87	01010111
88	88	01011000
89	89	01011001
90	90	01011010
91	91	01011011
92	92	01011100
93	93	01011101
94	94	01011110
95	95	01011111
96	96	01100000
97	97	01100001
98	98	01100010
99	99	01100011
100	100	01100100
101	101	01100101
102	102	01100110
103	103	01100111
104	104	01101000
105	105	01101001
106	106	01101010
107	107	01101011
108	108	01101100
109	109	01101101
110	110	01101110
111	111	01101111
112	112	01110000
113	113	01110001
114	114	01110010
115	115	01110011
116	116	01110100
117	117	01110101
118	118	01110110
119	119	01110111
120	120	01111000
121	121	01111001
122	122	01111010
123	123	01111011
124	124	01111100
125	125	01111101
126	126	01111110
127	127	01111111

# Katakana Display Characters

These characters can be displayed on the screen of Model 216, 217, 220, 226, and 236 computers, and on Series 300 computers using a 98546 Display Compatibility Interface.

ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary		Dec	Binary		Dec	Binary		Dec	Binary
ㇿ	128	10000000	ㇿ	160	10100000	ㇿ	192	11000000	ㇿ	224	11100000
ㇿ	129	10000001	ㇿ	161	10100001	ㇿ	193	11000001	ㇿ	225	11100001
ㇿ	130	10000010	ㇿ	162	10100010	ㇿ	194	11000010	ㇿ	226	11100010
ㇿ	131	10000011	ㇿ	163	10100011	ㇿ	195	11000011	ㇿ	227	11100011
ㇿ	132	10000100	ㇿ	164	10100100	ㇿ	196	11000100	ㇿ	228	11100100
ㇿ	133	10000101	ㇿ	165	10100101	ㇿ	197	11000101	ㇿ	229	11100101
ㇿ	134	10000110	ㇿ	166	10100110	ㇿ	198	11000110	ㇿ	230	11100110
ㇿ	135	10000111	ㇿ	167	10100111	ㇿ	199	11000111	ㇿ	231	11100111
ㇿ	136	10001000	ㇿ	168	10101000	ㇿ	200	11001000	ㇿ	232	11101000
ㇿ	137	10001001	ㇿ	169	10101001	ㇿ	201	11001001	ㇿ	233	11101001
ㇿ	138	10001010	ㇿ	170	10101010	ㇿ	202	11001010	ㇿ	234	11101010
ㇿ	139	10001011	ㇿ	171	10101011	ㇿ	203	11001011	ㇿ	235	11101011
ㇿ	140	10001100	ㇿ	172	10101100	ㇿ	204	11001100	ㇿ	236	11101100
ㇿ	141	10001101	ㇿ	173	10101101	ㇿ	205	11001101	ㇿ	237	11101101
ㇿ	142	10001110	ㇿ	174	10101110	ㇿ	206	11001110	ㇿ	238	11101110
ㇿ	143	10001111	ㇿ	175	10101111	ㇿ	207	11001111	ㇿ	239	11101111
ㇿ	144	10010000	ㇿ	176	10110000	ㇿ	208	11010000	ㇿ	240	11110000
ㇿ	145	10010001	ㇿ	177	10110001	ㇿ	209	11010001	ㇿ	241	11110001
ㇿ	146	10010010	ㇿ	178	10110010	ㇿ	210	11010010	ㇿ	242	11110010
ㇿ	147	10010011	ㇿ	179	10110011	ㇿ	211	11010011	ㇿ	243	11110011
ㇿ	148	10010100	ㇿ	180	10110100	ㇿ	212	11010100	ㇿ	244	11110100
ㇿ	149	10010101	ㇿ	181	10110101	ㇿ	213	11010101	ㇿ	245	11110101
ㇿ	150	10010110	ㇿ	182	10110110	ㇿ	214	11010110	ㇿ	246	11110110
ㇿ	151	10010111	ㇿ	183	10110111	ㇿ	215	11010111	ㇿ	247	11110111
ㇿ	152	10011000	ㇿ	184	10111000	ㇿ	216	11011000	ㇿ	248	11111000
ㇿ	153	10011001	ㇿ	185	10111001	ㇿ	217	11011001	ㇿ	249	11111001
ㇿ	154	10011010	ㇿ	186	10111010	ㇿ	218	11011010	ㇿ	250	11111010
ㇿ	155	10011011	ㇿ	187	10111011	ㇿ	219	11011011	ㇿ	251	11111011
ㇿ	156	10011100	ㇿ	188	10111100	ㇿ	220	11011100	ㇿ	252	11111100
ㇿ	157	10011101	ㇿ	189	10111101	ㇿ	221	11011101	ㇿ	253	11111101
ㇿ	158	10011110	ㇿ	190	10111110	ㇿ	222	11011110	ㇿ	254	11111110
ㇿ	159	10011111	ㇿ	191	10111111	ㇿ	223	11011111	ㇿ	255	11111111

Note 1: Characters 128 through 135 are "monochrome highlights" (see preceding table).

Note 2: Characters 136 through 143 are "color highlights" (see preceding table).

Note 3: Characters 144 through 159 are ignored by PRINT and DISP statements.



# Katakana Display Characters

These characters can be displayed on the Model 237 and on all Series 300 bit-mapped alpha displays.

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
0	0	00000000
1	1	00000001
2	2	00000010
3	3	00000011
4	4	00000100
5	5	00000101
6	6	00000110
7	7	00000111
8	8	00001000
9	9	00001001
10	10	00001010
11	11	00001011
12	12	00001100
13	13	00001101
14	14	00001110
15	15	00001111
16	16	00010000
17	17	00010001
18	18	00010010
19	19	00010011
20	20	00010100
21	21	00010101
22	22	00010110
23	23	00010111
24	24	00011000
25	25	00011001
26	26	00011010
27	27	00011011
28	28	00011100
29	29	00011101
30	30	00011110
31	31	00011111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
32	32	00100000
33	33	00100001
34	34	00100010
35	35	00100011
36	36	00100100
37	37	00100101
38	38	00100110
39	39	00100111
40	40	00101000
41	41	00101001
42	42	00101010
43	43	00101011
44	44	00101100
45	45	00101101
46	46	00101110
47	47	00101111
48	48	00110000
49	49	00110001
50	50	00110010
51	51	00110011
52	52	00110100
53	53	00110101
54	54	00110110
55	55	00110111
56	56	00111000
57	57	00111001
58	58	00111010
59	59	00111011
60	60	00111100
61	61	00111101
62	62	00111110
63	63	00111111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
64	64	01000000
65	65	01000001
66	66	01000010
67	67	01000011
68	68	01000100
69	69	01000101
70	70	01000110
71	71	01000111
72	72	01001000
73	73	01001001
74	74	01001010
75	75	01001011
76	76	01001100
77	77	01001101
78	78	01001110
79	79	01001111
80	80	01010000
81	81	01010001
82	82	01010010
83	83	01010011
84	84	01010100
85	85	01010101
86	86	01010110
87	87	01010111
88	88	01011000
89	89	01011001
90	90	01011010
91	91	01011011
92	92	01011100
93	93	01011101
94	94	01011110
95	95	01011111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
96	96	01100000
97	97	01100001
98	98	01100010
99	99	01100011
100	100	01100100
101	101	01100101
102	102	01100110
103	103	01100111
104	104	01101000
105	105	01101001
106	106	01101010
107	107	01101011
108	108	01101100
109	109	01101101
110	110	01101110
111	111	01101111
112	112	01110000
113	113	01110001
114	114	01110010
115	115	01110011
116	116	01110100
117	117	01110101
118	118	01110110
119	119	01110111
120	120	01111000
121	121	01111001
122	122	01111010
123	123	01111011
124	124	01111100
125	125	01111101
126	126	01111110
127	127	01111111



# Katakana Display Characters

These characters can be displayed on the Model 237 and on all Series 300 bit-mapped alpha displays.

ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary		Dec	Binary		Dec	Binary		Dec	Binary
カ	128	10000000	ハ	160	10100000	タ	192	11000000	エ	224	11100000
キ	129	10000001	ヒ	161	10100001	チ	193	11000001	エ	225	11100001
ク	130	10000010	フ	162	10100010	ツ	194	11000010	エ	226	11100010
ケ	131	10000011	ジュ	163	10100011	テ	195	11000011	エ	227	11100011
コ	132	10000100	ヘ	164	10100100	ト	196	11000100	エ	228	11100100
ク	133	10000101	マ	165	10100101	ナ	197	11000101	エ	229	11100101
カ	134	10000110	フ	166	10100110	ニ	198	11000110	エ	230	11100110
キ	135	10000111	ア	167	10100111	ヌ	199	11000111	エ	231	11100111
コ	136	10001000	イ	168	10101000	ネ	200	11001000	エ	232	11101000
カ	137	10001001	ウ	169	10101001	ノ	201	11001001	エ	233	11101001
キ	138	10001010	エ	170	10101010	ハ	202	11001010	エ	234	11101010
ク	139	10001011	オ	171	10101011	ヒ	203	11001011	エ	235	11101011
ケ	140	10001100	ヤ	172	10101100	フ	204	11001100	エ	236	11101100
コ	141	10001101	ユ	173	10101101	ヘ	205	11001101	エ	237	11101101
カ	142	10001110	ヨ	174	10101110	ホ	206	11001110	エ	238	11101110
キ	143	10001111	ツ	175	10101111	マ	207	11001111	エ	239	11101111
コ	144	10010000	ー	176	10110000	ミ	208	11010000	エ	240	11110000
カ	145	10010001	ア	177	10110001	ム	209	11010001	エ	241	11110001
キ	146	10010010	イ	178	10110010	メ	210	11010010	エ	242	11110010
ク	147	10010011	ウ	179	10110011	モ	211	11010011	エ	243	11110011
ケ	148	10010100	エ	180	10110100	ヤ	212	11010100	エ	244	11110100
コ	149	10010101	オ	181	10110101	ユ	213	11010101	エ	245	11110101
カ	150	10010110	カ	182	10110110	ヨ	214	11010110	エ	246	11110110
キ	151	10010111	キ	183	10110111	ラ	215	11010111	エ	247	11110111
コ	152	10011000	ク	184	10111000	リ	216	11011000	エ	248	11111000
カ	153	10011001	ケ	185	10111001	ル	217	11011001	エ	249	11111001
キ	154	10011010	コ	186	10111010	レ	218	11011010	エ	250	11111010
ク	155	10011011	サ	187	10111011	ロ	219	11011011	エ	251	11111011
ケ	156	10011100	シ	188	10111100	ワ	220	11011100	エ	252	11111100
コ	157	10011101	ス	189	10111101	ン	221	11011101	エ	253	11111101
カ	158	10011110	セ	190	10111110	ノ	222	11011110	エ	254	11111110
キ	159	10011111	リ	191	10111111	ハ	223	11011111	エ	255	11111111

Note 1: Characters 128 through 135 are "monochrome highlights" (see preceding table).

Note 2: Characters 136 through 143 are "color highlights" (see preceding table).

Note 3: Characters 144 through 159 are ignored by PRINT and DISP statements.

# Master Reset Table

	Power On	SCRATCH A	SCRATCH B	SCRATCH C	RESET	Note 2 END/ STOP	LOAD	LOAD &Go	GET	GET &Go	LOADSUB	Main Prerun	SUB Entry	SUB Exit
<b>CRT</b>														
CRT DISP Line	Clear	Clear	—	—	Clear	—	—	—	—	—	—	—	—	—
CRT Display Functions	Off	Off	—	—	—	—	—	—	—	—	—	—	—	—
CRT Message Line	Ready	Clear	Clear	Clear	Reset	—	—	—	—	—	—	Clear	—	—
CRT Input Line (Note 6)	Clear	Clear	Clear	—	Clear	—	—	—	—	—	—	—	—	—
CRT Printout Area	Clear	Clear	—	—	—	—	—	—	—	—	—	—	—	—
CRT Print Position (TABXY)	1,1	1,1	—	—	Note 15	—	—	—	—	—	—	—	—	—
ALPHA ON/OFF (Note 3)	On	On	On	On	On	On	—	—	—	—	—	—	—	—
<b>KEYBOARD</b>														
Keyboard Recall Buffer	Clear	—	—	—	—	—	—	—	—	—	—	—	—	—
Keyboard Result Buffer	Empty	Empty	—	—	—	—	—	—	—	—	—	—	—	—
Keyboard Knob Mode	↑	↑	↑	↑	↑	—	—	—	—	—	—	—	—	—
Tabs On Input Line	None	None	—	—	—	—	—	—	—	—	—	—	—	—
Typing Aid Labels	Note 16	Note 16	—	—	—	—	—	—	—	—	—	—	—	—
Keyboard Katakana Mode	Off	Off	Off	—	Off	—	—	—	—	—	—	—	—	—
SUSPEND INTERACTIVE	Off	Off	Off	Off	Off	Off	Off	Off	Off	Off	—	Off	—	—
<b>PRINTING</b>														
Print column	1	1	—	—	1	—	—	—	—	—	—	—	—	—
PRINTALL	Off	Off	—	—	Off	—	—	—	—	—	—	—	—	—
PRINTALL IS	1	1	—	—	—	—	—	—	—	—	—	—	—	—
PRINTER IS	1	1	—	—	—	—	—	—	—	—	—	—	—	—
<b>ENVIRONMENTS &amp; VARIABLES</b>														
Allocated Variables	None	None	None	None	Note 1	Note 1	None	None	None	None	—	None	None	Pre-ent
Normal Variables	None	None	None	None	—	—	None	None	None	None	—	Note 11	Note 11	Pre-ent
COM Variables	None	None	—	None	—	—	—	Note 9	—	Note 9	—	—	—	—
OPTION BASE	0	0	0	—	—	—	—	Note 9	—	Note 9	—	Note 9	Note 9	Pre-ent
I/O Path Names	None	Closed	Closed	Closed	None	Closed	Closed	Closed	Closed	Closed	—	Closed	—	sub clsd
I/O Path Names in COM	None	Closed	—	Closed	None	—	Note 10	Note 10	Note 10	Note 10	—	—	—	—
Keyboard Variable Access	No	No	No	No	Main	Main	No	In cnt.	No	In cnt.	In cnt.	Main	SUB	Pre-ent
BASIC Program Lines	None	None	None	—	—	—	Note 4	Note 4	Note 4	Note 4	Note 4	—	—	—
BASIC Program Environment	Main	Main	Main	Main	Main	Main	Main	Main	Main	Main	—	Main	SUB	Pre-ent
Normal Binary Programs	None	None	—	—	—	—	Note 5	Note 5	—	—	—	—	—	—
SUB Stack	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	Push	Pop
NPAR	0	0	0	0	0	0	0	0	0	0	—	0	Actual	Pre-ent
CONTINUE Allowed	No	No	No	No	No	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes
<b>ON &lt;event&gt; ACTIONS</b>														
ON <event> Log	Empty	Empty	Empty	Empty	Empty	Empty	Empty	Empty	Empty	Empty	—	Empty	Note 8	Note 8
System Priority	0	0	0	0	0	0	0	0	0	0	—	0	Note 7	Pre-ent
ON KEY Labels	None	None	None	None	None	None	None	None	None	None	—	None	—	Pre-ent
ENABLE/DISABLE	Enable	Enable	Enable	Enable	Enable	Enable	Enable	Enable	Enable	Enable	—	Enable	—	—
KNOBX & KNOBY	0	0	0	0	0	0	0	0	0	0	—	0	—	—

	Power On	SCRATCH A	SCRATCH	SCRATCH C	RESET	Note 2 END/ STOP	LOAD	LOAD &Go	GET	GET &Go	LOADSUB	Main Prerun	SUB Entry	SUB Exit
MISC.														
GOSUB Stack	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	Local	Pre-ent
TIMEDATE	Note 14	—	—	—	—	—	—	—	—	—	—	—	—	—
ERRL, ERRN, and ERRDS	0	0	—	—	—	—	—	0	—	0	—	0	—	—
ERRM\$	Null	Null	—	—	—	—	—	Null	—	Null	—	Null	—	—
DATA Pointer	None	None	None	None	None	None	None	1st main	None	1st main	—	1st main	1st sub	Pre-ent
LEXICAL ORDER IS	Stand.	Stand.	—	—	—	—	—	—	—	—	—	—	—	—
MASS STORAGE IS	Note 12	Note 12	—	—	—	—	—	—	—	—	—	—	—	—
CHECKREAD ON/OFF	Off	Off	—	—	—	—	—	—	—	—	—	—	—	—
Angle Mode	RAD	RAD	RAD	RAD	—	—	RAD	RAD	RAD	RAD	—	RAD	—	Pre-ent
Random Number Seed	Note 13	Note 13	Note 13	—	—	—	—	Note 13	—	Note 13	—	Note 13	—	—
DET	0	0	0	—	—	—	—	—	—	—	—	0	—	—
TRANSFER	None	Aborts	Note 17	Waits	Aborts	Waits	None	Note 18	None	Waits	—	None	—	Note 19
TRACE ALL	Off	Off	Off	—	—	—	—	—	—	—	—	—	—	—

— = Unchanged

Pre-ent = As existed previous to entry into the subprogram.

In cnt. = Access to variables in current context only.

1st main = Pointer set to first DATA statement in main program.

1st sub = Pointer set to first DATA statement in subprogram.

sub clsd = All local I/O path names are closed at subexit.

Waits = Operation waits until TRANSFER completes.

Note 1: Only those allocated in the main program are available.

Note 2: Pressing the STOP key is identical in function to executing STOP. Editing or altering a paused program causes the program to go into the stopped state.

Note 3: Alpha is turned on automatically by typing on the input line, by writing to the display line, or by an output to the message line.

Note 4: Modified according to the statement or command parameters and file contents.

Note 5: Any new binary programs in the file are loaded.

Note 6: Includes cursor position, INS CHR mode, ANY CHAR sequence state, but not tabs, auto-repeat rate, and auto-repeat delay. (These last three are defaulted only at SCRATCH A and Power On.)

Note 7: The system priority changes at SUB entry if the subroutine was invoked by an ON <event> CALL.

Note 8: See the appropriate keyword.

Note 9: As specified by the new environment or program.

Note 10: A COM mismatch between programs will close I/O path names. If I/O path names exist in a labeled COM, and a LOAD or GET brings in a program which does not contain that labeled COM, those I/O path names are closed.

Note 11: Numeric variables are set to 0, and string lengths are set to 0.

Note 12: The default mass storage device is INTERNAL (the right-hand drive) on the 9826 and 9836. See the 9816 Installation Manual for information on its default mass storage device.



## Further Comments

Note 13: The default random number seed is  $\text{INT}(\text{PI} \times (2^{31} - 2)/180)$ . This is equal to 37 480 660.

Note 14: The default TIMEDATE is 2.086 629 12 E + 11 (midnight March 1, 1900, Julian time).

Note 15: After a RESET, the CRT print position is in column one of the next line below the print position before the RESET.

Note 16: Typing aid labels are displayed unless a program is in the RUN state.

Note 17: Operation waits until TRANSFER completes unless both I/O path names are in COM.

Note 18: Operation waits until TRANSFER completes unless both I/O path names are in a COM area preserved during the LOAD.

Note 19: Operation waits until TRANSFER completes if the TRANSFER uses a local I/O path name.

The PAUSE key, the programmed PAUSE statement, and executing PAUSE from the keyboard all have identical effects. The only permanent effects of the sequence "PAUSE...CONTINUE" on a running program are:

1. Delay in execution.
2. Second and subsequent interrupt events of a given type are ignored.
3. INPUT, LINPUT, and ENTER 2 statements will be restarted.
4. ON KEY and ON KNOB are temporarily deactivated (i.e. not logged or executed) during the pause.
5. A TRANSFER may complete during the pause, causing ON EOT to be serviced at the next end-of-line.

Fatal program errors (i.e. those not trapped by ON ERROR) have the following effects:

- a PAUSE
- a beep
- an error message in the message line
- setting the values of the ERR1, the ERRN, and possibly the ERRDS functions
- setting the default EDIT line number to the number of the line in which the error occurred.

Autostart is equivalent to: Power On, LOAD "AUTOST", RUN.

CLR IO terminates ENTER and OUTPUT on all interfaces, handshake setup operations, HP-IB control operations, DISP, ENTER from CRT or keyboard, CAT, LIST, external plotter output, and output to the PRINTER IS, PRINTALL IS, and DUMP DEVICE IS devices when they are external. CLR IO does not terminate CONTROL, STATUS, READIO, WRITEIO, TRANSFER, real-time clock operations, mass storage operations (other than CAT), OUTPUT 2 (keyboard), or message line output.

CLR IO clears any pending closure key action.

If CLR IO is used to abort a DUMP GRAPHICS to an external device, the external device may be in the middle of an escape-code sequence. Thus, it might be counting characters to determine when to return to normal mode (from graphics mode). This means that a subsequent I/O operation to the same device may yield "strange" results. Handling this situation is the responsibility of the user and is beyond the scope of the firmware provided with the product. Sending 75 ASCII nulls is one way to "clear" the 9876 Graphics Printer.



# Graphic Reset Table

	Power On	SCRATCH A	SCRATCH B	SCRATCH C	RESET	Note 2 END/ STOP	GINIT	Main Prerun
PLOTTER IS	CRT	CRT	—	—	CRT	—	CRT	—
Graphics Memory	Clear	Clear	—	—	Note 1	—	Note 1	—
VIEWPORT	hrd clip	hrd clip	—	—	hrd clip	—	hrd clip	—
X and Y Scaling (unit of measure)	GDU	GDU	—	—	GDU	—	GDU	—
Soft Clip	hrd clip	hrd clip	—	—	hrd clip	—	hrd clip	—
Current Clip	hrd clip	hrd clip	—	—	hrd clip	—	hrd clip	—
CLIP ON/OFF	Off	Off	—	—	Off	—	Off	—
PIVOT	0	0	—	—	0	—	0	—
AREA PEN	1	1	—	—	1	—	1	—
PEN	1	1	—	—	1	—	1	—
LINE TYPE	1.5	1.5	—	—	1.5	—	1.5	—
Pen Position	0.0	0.0	—	—	0.0	—	0.0	—
LORG	1	1	—	—	1	—	1	—
CSIZE	5.6	5.6	—	—	5.6	—	5.6	—
LDIR	0	0	—	—	0	—	0	—
PDIR	0	0	—	—	0	—	0	—
GRAPHICS ON/OFF	Off	Off	—	—	—	—	—	—
ALPHA ON/OFF (Note 3)	On	On	On	On	On	On	—	—
DUMP DEVICE IS	701	701	—	—	—	—	—	—
GRAPHICS INPUT IS	None	None	—	—	None	—	None	—
TRACK ... ON/OFF	Off	Off	—	—	Off	—	Off	—
Color Map (Note 4)	Off	Off	—	—	Note 5	—	Note 5	—
Drawing Mode	Norm	Norm	—	—	Norm	—	Norm	—

— = Unchanged

hrd clip = The default hard clip boundaries of the CRT.

Note 1: Although RESET leaves the graphics memory unchanged, it will be cleared upon execution of the next graphics statement that sets a default plotter following the RESET.

Note 2: Pressing the STOP key is identical to executing STOP. Altering a paused program causes the program to go into the stopped state.

Note 3: Alpha is turned on automatically by typing on the input line, by writing to the display line, or by an output to the message line.

Note 4: With color map off, 8 standard colors are available. With color map on, 16 user-defined colors are available. See PLOTTER IS.

Note 5: Although the color map remains unchanged, it is changed if a graphics statement selects the device as a default plotter.

# Interface Reset Table

	Power On	SCRATCH A	SCRATCH B	BASIC RESET	Note 5 END/ STOP	LOAD	GET	Note 6 Reset Cmd	Main Prerun	SUB Entry	SUB Exit	CLR I/O
<b>GPIO Card</b>												
Interrupt Enable Bit	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Active Timeout Counter	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	—	—	—
Enable Interrupt Mask	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Hardware Reset of Card (PRESET)	Reset	Note 1	Note 1	Reset	Note 1	Note 1	Note 1	Reset	Note 1	—	—	Note 1
PSTS Error Flag	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
<b>RS-232 Card</b>												
Interrupt Enable Bit	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Active Timeout Counter	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	—	—	—
Enable Interrupt Mask	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Hardware Reset of Card	Reset	Reset	—	Reset	—	—	—	Reset	—	—	—	—
Data Rate/Character Format	Switch	Switch	—	—	—	—	—	—	—	—	—	—
RTS-DTR Latch	Clear	Clear	—	—	—	—	—	Clear	—	—	—	—
Request to Send Line	Clear	Clear	—	Clear	—	—	—	Clear	—	—	—	Note 2
Data Terminal Ready Line	Clear	Clear	—	Clear	—	—	—	Clear	—	—	—	Note 2
Line Status Register	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	Clear
Modem Status Register	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	Clear
Data-In Buffer	Empty	Empty	Empty	Empty	Empty	Empty	Empty	Empty	Empty	—	—	Empty
Error-Pend. Flag	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	Clear
<b>HP-IB</b>												
Interrupt Enable Bit	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Active Timeout Counter	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	—	—	—
Interrupt Enable Mask	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
User Interrupt Status	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Serial Poll Register	Clear	Clear	—	Clear	—	—	—	Clear	—	—	—	—
Parallel Poll Register	Clear	Clear	—	Clear	—	—	—	Clear	—	—	—	—
My Address Register	Note 4	Note 4	—	—	—	—	—	—	—	—	—	—
IFC Sent	Note 3	Note 3	—	Note 3	—	—	—	Note 3	—	—	—	—
REN Set True	Note 3	Note 3	—	Note 3	—	—	—	Note 3	—	—	—	—
<b>Data Communications</b>												
Interrupt Enable Bit	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Active Timeout Counter	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	—	—	—
Interrupt Enable Mask	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Hardware Reset of Card	Reset	Note 7	—	Reset	—	—	—	Note 7	—	—	—	—
Line State	Dscon	Dscon	—	Dscon	—	—	—	Dscon	—	—	—	—
Data Buffers	Empty	Empty	—	Empty	—	—	—	Empty	—	—	—	—
Protocol Selection (Async or Data Link)	Switch	Note 8	—	Switch	—	—	—	Note 8	—	—	—	—
Protocol Options	Switch	Switch	—	Switch	—	—	—	Switch	—	—	—	—

	Power On	SCRATCH A	SCRATCH	BASIC RESET	Note 5 END/ STOP	LOAD	GET	Note 6 Reset Cmd	Main Prerun	SUB Entry	SUB Exit	CLR I/O
<b>BCD Card</b>												
Interrupt Enable Bit	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Active Timeout Counter	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	—	—	—
Interrupt Enable Mask	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Hardware Reset of Card	Reset	Note 1	Note 1	Note 1	Note 1	Note 1	Note 1	Reset	Note 1	—	—	Note 1
Rewind Driver	Rwd	Rwd	Rwd	Rwd	Rwd	Rwd	Rwd	Rwd	Rwd	—	—	Rwd
BCD/Binary Mode	Switch	Switch	—	—	—	—	—	—	—	—	—	—
<b>EPROM Programmer</b>												
Hardware Reset of Card	Reset	Reset	—	Reset	—	—	—	Reset	—	—	—	—
Programming Time Register	Clear	Clear	—	—	—	—	—	Clear	—	—	—	—
Target Address Register	Clear	Clear	—	—	—	—	—	Clear	—	—	—	—

— = Unchanged

Swch = Set according to the switches on the interface card

Dscon = A disconnect is performed

Note 1: Reset only if card is not ready.

Note 2: Cleared only if corresponding modem control line is not set.

Note 3: Sent only if System Controller.

Note 4: If System Controller and Active Controller, address is set to 21. Otherwise, it is set to 20.

Note 5: Pressing the STOP key is identical in function to executing STOP or END. Editing or altering a paused program causes the program to go into the stopped state.

Note 6: Caused by sending a non-zero value to CONTROL register 0.

Note 7: This is a "soft reset," which does not include an interface self-test or a reconfiguration of protocol.

Note 8: Set according to the value used in the most recent CONTROL statement directed to Register 3. If there has been no CONTROL 3 statement, the switch settings are used.



# Selected High-Precision Metric Conversion Factors

English Units	Metric Units	To convert from English to Metric, multiply by:	To convert from Metric to English, multiply by:
<b>Length</b>			
mil	micrometre (micron)	$2.54 \times 10^1 \star$	$3.937\ 007\ 874 \times 10^{-2}$
inch	millimetre	$2.54 \times 10^1 \star$	$3.937\ 007\ 874 \times 10^{-2}$
foot	metre †	$3.048 \times 10^{-1} \star$	3.280 839 895
mile (intl.)	kilometre	1.609 344 $\star$	$6.213\ 711\ 922 \times 10^{-1}$
<b>Area</b>			
inch <sup>2</sup>	millimetre <sup>2</sup>	$6.451\ 6 \times 10^2 \star$	$1.550\ 003\ 100 \times 10^{-3}$
foot <sup>2</sup>	metre <sup>2</sup>	$9.290\ 304 \times 10^{-2} \star$	$1.076\ 391\ 042 \times 10^1$
mile <sup>2</sup>	kilometre <sup>2</sup>	2.589 988 110	$3.861\ 021\ 585 \times 10^{-1}$
acre (U.S. survey)	hectare	$4.046\ 873 \times 10^{-1}$	2.471 044
<b>Volume</b>			
inches <sup>3</sup>	millimetres <sup>3</sup>	$1.638\ 706\ 4 \times 10^4 \star$	$6.102\ 374\ 409 \times 10^{-5}$
feet <sup>3</sup>	metres <sup>3</sup>	$2.831\ 684\ 659 \times 10^{-2} \star$	$3.531\ 466\ 672 \times 10^1$
ounces (U.S. fluid)	centimetres <sup>3</sup>	$2.957\ 353 \times 10^1$	$3.381\ 402 \times 10^{-2}$
gallon (U.S. fluid)	litre ‡	3.785 412	$2.641\ 721 \times 10^{-1}$
<b>Mass</b>			
pound (avdp.)	kilogram	$4.535\ 923\ 7 \times 10^{-1} \star$	2.204 622 622
ton (short)	ton (metric)	$9.071\ 847\ 4 \times 10^{-1} \star$	1.102 311 311
<b>Force</b>			
ounce (force)	dyne	$2.780\ 138\ 510 \times 10^4$	$3.596\ 943\ 090 \times 10^{-5}$
pound (force)	newton	4.448 221 615	$2.248\ 089\ 431 \times 10^{-1}$
<b>Pressure</b>			
psi	pascal	$6.894\ 757\ 293 \times 10^3$	$1.450\ 377\ 377 \times 10^{-4}$
inches of Hg (at 32°F)	millibar	$3.386\ 4 \times 10^1$	$2.952\ 9 \times 10^{-2}$
<b>Energy</b>			
BTU (IST)	Calorie (kg, thermochem.)	$2.521\ 644\ 007 \times 10^{-1}$	3.965 666 831
BTU (IST)	watt-hour	$2.930\ 710\ 702 \times 10^{-1}$	3.412 141 633
BTU (IST)	joule §	$1.055\ 055\ 853 \times 10^3$	$9.478\ 171\ 203 \times 10^{-4}$
ft•lb	joule	1.355 817 948	$7.375\ 621\ 493 \times 10^{-1}$
<b>Power</b>			
BTU (IST)/hr	watt	$2.930\ 710\ 702 \times 10^{-1}$	3.412 141 633
horsepower (mechanical)	watt	$7.456\ 998\ 716 \times 10^2$	$1.341\ 022\ 090 \times 10^{-3}$
horsepower (electric)	watt	$7.46 \times 10^2 \star$	$1.340\ 482\ 574 \times 10^{-3}$
ft•lb/s	watt	1.355 817 948	$7.375\ 621\ 493 \times 10^{-1}$
<b>Temperature</b>			
°Rankine	kelvin	1.8 $\star$	$5.555\ 555\ 556 \times 10^{-1}$
°Fahrenheit	°Celsius	$^{\circ}\text{C} = (^{\circ}\text{F} - 32) / 1.8 \star$	$^{\circ}\text{F} = (^{\circ}\text{C} \times 1.8) + 32 \star$

☆ Exact conversion

† Conversion redefined in 1959

‡ Conversion redefined in 1964

§ Conversion redefined in 1956

**Note:** The preferred metric unit for force is the newton; for pressure, the pascal; and for energy, the joule.

## Sources

American Society for Testing and Materials (ASTM), "Standard for Metric Practice". Reprinted from Annual Book of ASTM Standards.

U.S. Department of Commerce, National Bureau of Standards, "NBS Guidelines for the Use of the Metric System". Reprinted from Dimensions/NBS. (October 1977).

Prefix	Symbol	Multiplier
exa	E	$10^{18}$
peta	P	$10^{15}$
tera	T	$10^{12}$
giga	G	$10^9$
mega	M	$10^6$
kilo	k	$10^3$
hecto	h	$10^2$
deka	da	$10^1$

Prefix	Symbol	Multiplier
deci	d	$10^{-1}$
centi	c	$10^{-2}$
milli	m	$10^{-3}$
micro	$\mu$	$10^{-6}$
nano	n	$10^{-9}$
pico	p	$10^{-12}$
femto	f	$10^{-15}$
atto	a	$10^{-18}$





# Index

---

## a

Abort message .....	12-23
ABORT statement .....	12-12, 12-16, 14-18
ABORTIO statement .....	9-20
Above-Screen Lines .....	10-23
Absolute Positioners .....	16-27
Active controller .....	12-32
Additional Interface Functions .....	2-6
Address, primary .....	3-5
Addressed to listen, HP-IB .....	12-9
Addressed to talk, HP-IB .....	12-9
Addressing multiple listeners on the HP-IB bus .....	12-11
Addressing, Non-Active HP-IB Controller .....	12-39
Addressing, Secondary .....	12-11
ALPHA HEIGHT statement .....	10-5, 10-6
ALPHA PEN statement .....	10-8, 10-32
ASCII and Non-ASCII Keys .....	11-3
ASCII Files .....	8-22
ASCII Representation of Integers .....	2-16
ASCII Representation of Real Numbers .....	2-17
ASCII representations .....	15-18
ASSIGN statement .....	3-7, 3-8, 3-9, 8-4, 9-6
ASSIGN Statements, Determining the Outcome of .....	8-18
Asynchronous Communication Protocol .....	14-3
Asynchronous Data Communication .....	13-2
Attention Line (ATN), HP-IB .....	12-50
Attribute, BYTE .....	8-6
Attribute control .....	3-12
Attribute, WORD .....	8-6
Attributes, Additional .....	8-6
Attributes, Changing Buffer .....	9-37
Attributes, FORMAT .....	8-2
Attributes, I/O Path .....	8-1
Attributes, Restoring the Default .....	8-5
Auto-repeat, keyboard .....	11-9

## **b**

Backplane, computer .....	2-3
Bar Code Reader, Using a .....	16-51
BASIC/DOS .....	1-1
BASIC/UX .....	1-1
BASIC/WS .....	1-1
Baud rate (RS-232C) .....	13-10
Baud Rate, RS-232C Handshake and .....	13-6
BCD Representation .....	15-28
BDAT Files .....	8-21
Binary Images .....	5-21
Binary images .....	4-18
Binary specifier .....	4-18
Bits and Bytes .....	2-12
Branch, initiating a .....	7-5
BREAK Message .....	13-16
Break received .....	13-14
Buffer Attributes, Changing .....	9-37
Buffer, Named .....	9-5
Buffer Pointers .....	9-8
Buffer Size Register .....	9-7
BUFFER statement .....	8-4, 9-5, 9-6
Buffer Status and Control Registers .....	9-38
Buffer, Unnamed .....	9-5
Buffer-Type Registers .....	9-7
Buffered I/O .....	9-32
Buffers, A Closer Look at .....	9-5
Buffers and Transfers, Overview of .....	9-2
Buffers, Creating Named .....	9-5
Buffers, Types of .....	9-5
Burst I/O mode .....	3-14, 9-31, 12-7, 15-25
Burst I/O mode during a TRANSFER .....	9-31
Burst transfers .....	9-12
Bus .....	2-2
BYTE Attribute .....	8-6
BYTE attribute .....	8-7, 8-11
Byte count .....	9-8

## **c**

Cable Options and Functions, Datacomm .....	14-20
Cable options, RS-232C .....	13-18

Caps Lock Mode .....	11-7
cat command .....	3-19
CDIAL function .....	16-26
Chapter Previews .....	1-3
Character conversions .....	15-31
Character Format and Parity, RS-232C .....	13-11
Character Format Definition, Datacomm .....	14-14
Character Format Parameters, RS-232C .....	13-7
Character Length (RS-232C) .....	13-7
Character specifier .....	4-17
Characters, Converting .....	8-11
Characters, Ignoring .....	5-20
Characters, Representing .....	2-14
Circuit Driver/Receiver Functions, Optional .....	13-20
Clear Lockout/Local message .....	12-22
Clear message .....	12-22
CLEAR SCREEN statement .....	10-5
CLEAR statement .....	12-12, 12-15
Clear to Send (CTS), RS-232C .....	13-6
Clearing the Screen .....	10-5
Closing I/O Path Names .....	3-9
Closure Keys .....	11-22
CMD secondary keyword .....	12-30
Color Enhancements .....	10-20
Comma separator .....	4-4
command line .....	3-16
commands:	
cat .....	3-19
tee .....	3-19
wc .....	3-18
who .....	3-18
Communicating with HP-IB devices .....	12-3
Computer As a Non-Active Controller on the HP-IB Bus .....	12-32
Computer backplane .....	2-3
Concurrency .....	9-27
Conditions, Interrupt .....	7-19
Configuration Switches .....	13-32
Configuring Parallel Poll Responses .....	12-19
Connecting processes with pipes .....	3-18
Control Characters .....	10-16
Control characters, generating .....	11-4, 11-5, 11-6
Control, Passing .....	12-34



CONTROL statement .....	6-3
Control-Character Functions .....	10-18
Controller address, HP-IB .....	12-32
Controller status, HP-IB .....	12-32
Controller's Address, Changing the HP-IB .....	12-34
CONVERT IN statement .....	8-14
CONVERT OUT statement .....	8-14
CONVERT statement .....	8-11
CONVERT...BY INDEX statement .....	8-11
CONVERT...BY PAIRS statement .....	8-13
Copying Data into the Destinations .....	2-23
Copying Data to the Destination .....	2-21
COUNT parameter .....	9-15
Creating windows .....	17-2
CRT STATUS and CONTROL Registers .....	10-37

## d

Data Carrier Detect (DCD or CD), RS-232C .....	13-6
Data Communication Equipment (DCE), RS-232C .....	13-18
Data Compatibility .....	2-6
Data, Entering .....	5-1
Data entry, RS-232C .....	13-13
Data Flow, Directing .....	3-1
Data Handshake .....	2-19
Data message .....	12-9, 12-22, 12-29
Data on the HP-IB bus, Sending .....	12-29
Data output, RS-232C .....	13-12
Data, Outputting .....	4-1
Data, Re-Directing .....	3-11
Data Representation Summary .....	8-24
Data Representations .....	2-12
DATA secondary keyword .....	12-30
Data Set Ready (DSR), RS-232C .....	13-6
Data Terminal Equipment (DTE), RS-232C .....	13-18
Data to the Keyboard, Sending .....	11-14
Data Transfers, RS-232C .....	13-12
Data Valid (DAV), HP-IB .....	12-50
Data-Representation Design Criteria .....	8-20
Datacomm adapter options and functions .....	14-20
Datacomm character format definition .....	14-14
Datacomm configuration for BASIC/UX .....	14-9

Datacomm connection .....	14-8
Datacomm Data Transfers Between Computer and Interface .....	14-4
Datacomm Error Detection and Program Recovery .....	14-19
Datacomm error recovery .....	14-18
Datacomm interface .....	2-9
Datacomm interface (with TRANSFER) .....	9-35
Datacomm interfaces .....	14-1
Datacomm line connection .....	14-15
Datacomm Line Timeouts .....	14-12
Datacomm Options for Async Communications .....	14-12
Datacomm Parity option:	
EVEN .....	14-3
NONE .....	14-3
ODD .....	14-3
Datacomm Programming .....	14-7
Datacomm Protocol and Link Operating Parameters .....	14-8
Datacomm Start bits .....	14-3
Datacomm Stop bits .....	14-3
Datacomm Time gap .....	14-3
Datacomm timeouts .....	14-12
DCE cable option .....	13-18
DCE cable options .....	13-20, 14-20
DCE Cable, RS-232C .....	13-20
DELAY statement .....	8-15
DELIM parameter .....	9-15
Delimiter Characters .....	9-15
Device Selectors .....	3-3
Digit specifier .....	4-14
DIGITIZE statement .....	16-25
Direct Connection Links, Datacomm .....	14-15
Direct Interface Access .....	6-12
Direct memory access .....	9-12, 9-32
Directing Data Flow .....	3-1
DISABLE EXT SIGNAL statement .....	7-22
Disabling the Cursor Character .....	10-32
DISP Line .....	10-31
Display features, overview of .....	10-3
Display Functions Mode .....	10-21
DISPLAY FUNCTIONS statement .....	10-21
Display interfaces .....	10-1
Display Line, Output Area and the .....	10-5
Display regions .....	10-4, 10-8

Display Regions Affected by Pen Color Statements .....	10-9
Display-Enhancement Characters .....	10-19
DMA transfers .....	9-12
DRS Modem Line, Programming the .....	13-17
DTE cable options .....	13-20, 14-20
DTE Cable, RS-232C .....	13-19

## e

Electrical and Mechanical Compatibility .....	2-5
Empty pointer .....	9-8
ENABLE EXT SIGNAL statement .....	7-22
ENABLE INTR, GPIO .....	15-33
ENABLE INTR statement .....	12-17
Enabling and setting up GPIO events .....	15-32
Enabling Local Control .....	12-14
Enabling the Insert Mode .....	10-33
END in Freefield OUTPUT .....	4-8
End or Identify Line (EOI) .....	12-51
END parameter .....	9-16
END with Data Communications Interfaces .....	4-26
END with HP-IB Interfaces .....	4-9, 4-26
END with OUTPUTs that Use Images .....	4-25
END with the Data Communications Interface .....	4-9
End-of-line (EOL) .....	4-3
End-of-line sequence .....	4-6, 8-1, 8-15
End-or-identify .....	5-12, 5-23
Enhanced Keyboard Control .....	11-27
ENTER and Buffers, OUTPUT and .....	9-13
ENTER images .....	4-21
ENTER statement .....	2-18, 2-22, 2-23, 3-2, 5-1, 5-14
ENTER USING statement .....	5-14
Entering Data .....	5-1
Entering Data from the Keyboard .....	11-11
Entering from the CRT .....	10-28
Entering String Data .....	5-8
Enters that Use Images .....	5-14
EOI Re-Definition .....	5-23
EOI Signal, Sending the .....	11-13
ERRL function .....	14-19
ERRN function .....	14-19
Error Detection and Program Recovery, Datacomm .....	14-19



Error Detection, RS-232C .....	13-4
Error Recovery, Datacomm .....	14-18
Error Reporting .....	9-29
Event-Initiated Branching .....	7-1
Events, Enabling Interrupt .....	7-15
Events, Logging and Servicing .....	7-6
Events, Servicing Pending .....	7-12
Events, Types of .....	7-1
Execution Speed .....	3-10
Explicitly close .....	3-9
Exponent specifier .....	4-14
External interrupt request .....	15-33

## f

Files, ASCII .....	8-22
Files, BDAT .....	8-21
Files, I/O Paths to .....	8-20
Fill pointer .....	9-8
filter programs .....	3-20
Firmware .....	2-18
FORMAT attributes .....	8-2
FORMAT Attributes, Assigning Default .....	8-4
FORMAT OFF statement .....	3-12, 8-2, 8-3
FORMAT ON statement .....	3-12, 8-2
FORMAT statement .....	8-2
Formatting, Transfer .....	9-13
Framing error (RS-232C) .....	13-4, 13-14
Free-Field ENTER Statements .....	5-11
Free-Field Enters .....	5-1
Free-field output .....	4-1, 4-2
Freefield OUTPUT, END in .....	4-8
Function Box .....	16-35
Function Box, Activating the .....	16-36
Function Box key presses, Trapping .....	16-38
Function Box Keys, Assigning Functions to .....	16-41

## g

GPIO burst I/O mode .....	15-25
GPIO control output lines, driving the .....	15-41
GPIO data handshake methods .....	15-5
GPIO data logic sence .....	15-5



GPIO data-in clock source .....	15-7
GPIO ENABLE INTR .....	15-33
GPIO events, enabling and setting up .....	15-32
GPIO Full Handshake Transfer .....	15-38
GPIO full-mode handshakes .....	15-8
GPIO handshake lines .....	15-6
GPIO handshake logic sence .....	15-6
GPIO handshake modes .....	15-6
GPIO hardware interrupt priority .....	15-5
GPIO interface .....	2-11, 15-1
GPIO interface configuration .....	15-4
GPIO interface reset .....	15-17
GPIO Interface Select Code .....	15-5
GPIO interface (with TRANSFER) .....	9-35
GPIO interrupt transfers .....	15-39
GPIO interrupts .....	15-32
GPIO optional peripheral status check .....	15-7
GPIO, Outputs and Enters through the .....	15-18
GPIO pulse-mode handshakes .....	15-11
GPIO ready interrupt transfers .....	15-39
GPIO special-purpose lines .....	15-41
GPIO statements that enter data bytes .....	15-20
GPIO statements that enter data words .....	15-23
GPIO statements that output data bytes .....	15-19
GPIO statements that output data words .....	15-22
GPIO STATUS and CONTROL Registers .....	15-44
GPIO status input lines, interrogating the .....	15-42
GPIO Timeouts .....	15-26
GPIO transfer design .....	15-37
GPIO, Types of Interrupt Events .....	15-32

## h

Handshake .....	12-9
Handshake and Baud Rate, RS-232C .....	13-6
Handshake, Data .....	2-19
Handshake, Datacomm .....	14-14
Handshake Lines, HP-IB .....	12-50
Hardware priority .....	7-10
HIL Devices, Re-Configuring .....	11-2
HIL SEND statement .....	16-4
HILBUF\$ function .....	16-5

HIL_ID program .....	16-8
HIL_ID program explanation .....	16-10
HP 13265 Modem .....	14-2
HP 13266A Current Loop Adapter .....	14-2
HP 92916A (Bar-Code Reader) .....	16-30
HP 98626 RS-232 Serial Interface .....	13-31
HP 98628 Serial Interface STATUS and CONTROL Registers .....	13-25
HP 98642 4-Channel Multiplexer .....	2-10
HP 98642 4-channel multiplexer .....	14-25
HP 98642 Serial Interface STATUS and CONTROL Registers .....	13-25
HP 98644 RS-232 Serial Interface .....	13-31
HP-HIL Device Characteristics .....	16-31
HP-HIL device preview .....	16-3
HP-HIL devices .....	16-22
HP-HIL Devices, Communicating with .....	16-31
HP-HIL Devices, Interaction Between Multiple .....	16-55
HP-HIL devices supported by the HIL Interface driver .....	16-6
HP-HIL ID Module Data, Interpreting .....	16-33
HP-HIL ID Modules, Note about Installing and Removing .....	16-34
HP-HIL initialization .....	16-2
HP-HIL Interface .....	16-1
HP-HIL Interface, Communicating through the .....	16-4
HP-HIL interface driver statements .....	16-4
HP-HIL Keyboards .....	16-23
HP-HIL Link, Identifying All Devices on the .....	16-8
HP-HIL, Other Devices .....	16-29
HP-HIL Security Device .....	16-28
HP-IB ABORT .....	12-12
HP-IB active controller .....	12-9, 12-32
HP-IB Address Commands and Codes .....	12-25
HP-IB addressed to listen .....	12-9
HP-IB addressed to talk .....	12-9
HP-IB attention line (ATN) .....	12-9, 12-50
HP-IB burst I/O mode .....	12-7
HP-IB Bus Activity, Aborting .....	12-16
HP-IB bus, Addressing multiple listeners on the .....	12-11
HP-IB Bus Commands and Codes .....	12-24
HP-IB Bus Management .....	12-12
HP-IB Bus Management, Advanced .....	12-21
HP-IB Bus Message Types .....	12-22
HP-IB Bus Messages, Explicit .....	12-27
HP-IB bus sequences .....	12-10

HP-IB Bus-Line States, Determining .....	12-53
HP-IB CLEAR .....	12-12
HP-IB Control Lines .....	12-49
HP-IB controller address .....	12-32
HP-IB controller status .....	12-32
HP-IB data movement .....	12-4
HP-IB Data Valid (DAV) .....	12-50
HP-IB Device Selectors .....	3-5, 12-3
HP-IB Devices, Clearing .....	12-15
HP-IB devices, Communicating with .....	12-3
HP-IB Devices, Polling .....	12-19
HP-IB Devices, Triggering .....	12-15
HP-IB ENABLE INTR .....	12-17
HP-IB end-or-identify line (EOI) .....	12-51
HP-IB Handshake Lines .....	12-50
HP-IB Installation and Verification .....	12-2
HP-IB interface .....	2-7, 12-1
HP-IB, Interface Clear Line (IFC) .....	12-51
HP-IB interface (with TRANSFER) .....	9-35
HP-IB Interface-State Information .....	12-45
HP-IB interlocking handshake .....	12-50
HP-IB Interrupts that Require Data Transfers, Servicing .....	12-46
HP-IB LOCAL .....	12-12
HP-IB LOCAL LOCKOUT .....	12-12
HP-IB Message Mnemonics .....	12-30
HP-IB messages .....	12-21
HP-IB NDAC holdoff .....	12-56
HP-IB Not Data Accepted (NDAC) .....	12-50
HP-IB Not Ready for Data (NRFD) .....	12-50
HP-IB ON INTR .....	12-17
HP-IB PPOLL .....	12-12
HP-IB PPOLL CONFIGURE .....	12-12
HP-IB PPOLL UNCONFIGURE .....	12-12
HP-IB REMOTE .....	12-12
HP-IB remote enable line (REN) .....	12-51
HP-IB Secondary Addressing .....	12-11
HP-IB select code .....	12-3
HP-IB SEND .....	12-12
HP-IB service request line (SRQ) .....	12-52
HP-IB service requests .....	12-16, 12-41
HP-IB SPOLL .....	12-12
HP-IB SRQ interrupts .....	12-17



HP-IB STATUS and CONTROL Registers .....	12-54
HP-IB Structure .....	12-8
HP-IB system controller .....	12-32
HP-IB TRIGGER .....	12-13
HP-IB:	
Abort message .....	12-23
Clear Lockout/Local message .....	12-22
Clear message .....	12-22
Data message .....	12-22
Local Lockout message .....	12-22
Local message .....	12-22
Pass Control message .....	12-22
Remote message .....	12-22
Service Request message .....	12-22
Status Bit message .....	12-22
Status Byte message .....	12-22
Trigger message .....	12-22
HP-UX pipes .....	3-15
HP 35723A (HP-HIL/Touchscreen) .....	16-28
HP 45911A (11 × 11 Graphics Tablet) .....	16-28
HP 46020/21A Keyboard .....	16-23
HP 46060A (HP-Mouse) .....	16-24
HP 46083A (Rotary Control Knob) .....	16-24
HP 46084A (HP-HIL ID Module) .....	16-28
HP 46086A (Function Box) .....	16-29
HP 46087A (A-size Digitizer) .....	16-28
HP 46088A (B-size Digitizer) .....	16-28
HP 46094A (HP-HIL/Quadrature Port) .....	16-24
HP 98203C Keyboard .....	16-23
HP 98622 Interface .....	15-1
HP 98626 and HP 98644 Card ID Register .....	13-31
HP 98626 Optional Driver Receiver Circuits .....	13-32
HP 98628 Data Communications Interface .....	14-1
HP 98642 Data Communications Interface .....	14-1
HP 98644 Card ID Register .....	13-34
HP 98644 Coverplate Connector .....	13-33

## i

Image Definitions During Outputs .....	4-13
Image OUTPUT .....	4-1
Image output .....	4-2



Image Re-Use .....	4-23, 5-26
Image Repeat Factors .....	4-22
Images .....	4-11, 5-15
Images, binary .....	4-18
Images, ENTER .....	4-21
Images, nested .....	4-24
Images, numeric .....	4-14
Images, Outputs that Use .....	4-10
Images, Special-Character .....	4-20
Images, string .....	4-17
Images, Terminating Enters that Use .....	5-22
Inbound and Outbound Transfers .....	9-2
Inbound Control Blocks, Datacomm .....	14-5
Inbound Datacomm Data Messages .....	14-6
Inbound pipes .....	3-21
Inbound transfer .....	9-2
Input .....	2-2
Integers, ASCII Representation of .....	2-16
Integers, Internal Representation of .....	2-15
Integers, Representing Signed .....	2-15
Integral Keyboard .....	16-23
Interactive Keyboard .....	11-31
Interface Access, Direct .....	6-12
Interface Clear Line (IFC), HP-IB .....	12-51
Interface Functions, Additional .....	2-6
Interface Interrupts .....	7-14
Interface locking .....	3-13
Interface, primary function of an .....	2-4
Interface ready .....	15-33
Interface Registers .....	6-2
Interface Reset, RS-232C .....	13-9
Interface Timeouts .....	7-20
Interfaces in the HP-UX Environment .....	3-13
Interfaces, select codes .....	3-4
Interfaces, Select Codes of Optional .....	3-4
Interfacing Concepts .....	2-1
Internal Representation of Integers .....	2-15
Internal Representation of Real Numbers .....	2-17
Internal representations .....	15-18
Interrupt Conditions .....	7-19
Interrupt service routine (ISR) .....	15-34
Interrupts and Timeouts .....	7-1

Interrupts, Non-Active HP-IB Controller .....	12-35
I/O .....	2-2
I/O, Applications of Unified .....	8-25
I/O, Concepts of Unified .....	8-19
I/O Examples .....	2-20
I/O Operations with String Variables .....	8-25
I/O Path Attributes .....	8-1
I/O Path Attributes, Specifying .....	8-5
I/O Path Benefits .....	3-10
I/O path name .....	3-6, 3-7, 6-9, 8-1, 9-7
I/O Path Names Assigned to a BDAT File .....	6-10
I/O Path Names Assigned to a Buffer .....	6-11
I/O Path Names Assigned to a Device .....	6-9
I/O Path Names Assigned to an ASCII File .....	6-9
I/O Path Names Assigned to an HP-UX File .....	6-10
I/O Path Names, Closing .....	3-9
I/O Path Names, Re-Assigning .....	3-8
I/O Path Names to Named Buffers, Assigning .....	9-6
I/O Path Names to Unnamed Buffers, Assigning .....	9-6
I/O Path Register Summary .....	6-9
I/O Path Registers .....	6-5
I/O Paths to Files .....	8-20
I/O Process .....	2-18
I/O Statements and Parameters .....	2-18
Item Separators .....	4-3, 5-2
Item Terminators .....	4-3, 5-2

## k

KBD\$ function .....	11-26, 11-27, 11-28, 16-23, 16-25
KBD LINE PEN statement .....	10-8
KBD Status and Control Registers .....	11-34
KEY LABELS ON/OFF statement .....	10-35
KEY LABELS PEN statement .....	10-8
Keyboard auto-repeat .....	11-9
Keyboard CAPS LOCK mode .....	11-7
Keyboard ENTER .....	11-11
Keyboard features .....	11-2
Keyboard, Interactive .....	11-31
Keyboard interfaces .....	11-1
Keyboard, Locking Out the .....	11-32
Keyboard Operating Modes .....	11-7

Keyboard OUTPUT .....	11-14
Keyboard types .....	11-1
Keystrokes, trapping .....	11-27
Knob Rotation .....	11-25
KNOBX function .....	11-26
KNOBY function .....	11-26

## I

Line connection, Datacomm .....	14-15
Line speed (baud rate), datacomm .....	14-13
Listing windows .....	17-4
LOADSUB ALL FROM .....	8-39
Local Control, Enabling .....	12-14
Local Lockout message .....	12-22
LOCAL LOCKOUT statement .....	12-12, 12-14
Local message .....	12-22
LOCAL statement .....	12-12
Locking an interface during a TRANSFER .....	9-31
Locking an interface to a process .....	12-6, 15-24
Locking, interface .....	3-13
Locking Out Local Control .....	12-14
Locking Out the Keyboard .....	11-32

## m

Manual Organization .....	1-2
Mechanical Compatibility, Electrical and .....	2-5
Modem Control Register, RS-232C .....	13-16
Modem Handshake Lines, RS-232C .....	13-16
Modem Line Handshaking, RS-232C .....	13-13
Modifiers, Statement-Termination .....	5-24
Monochrome enhancement characters .....	10-19
Mouse Keys .....	11-30
Moving windows .....	17-7
Multi-tasking environment .....	2-23
Multi-user environment .....	2-24
Multiple Termination Conditions .....	9-16
Multiplexer device selectors .....	3-5
Multiplexer, HP 98642 4-channel .....	14-25
Multiplexer interface .....	2-10



## n

Named buffer .....	9-5
Named Buffers, Assigning I/O Path Names to .....	9-6
Named Buffers, Creating .....	9-5
Named Buffers via Variable Names, Accessing .....	9-10
Named pipe .....	3-17, 3-22
Names, of variables (syntax of) .....	3-6
Names, string-variable .....	3-2
NDAC holdoff, HP-IB .....	12-56
Nested Images .....	4-24, 5-26
Non-Active HP-IB Controller Addressing .....	12-39
Non-Active HP-IB Controller Interrupts .....	12-35
Non-ASCII Keystrokes .....	11-14
Non-Repeatable Specifiers .....	5-26
Not Data Accepted (NDAC), HP-IB .....	12-50
Not Ready for Data (NRFD), HP-IB .....	12-50
Number builder .....	5-3
Numbers, Representing .....	2-13
Numeric Format, Standard .....	4-2
Numeric Images .....	4-14, 5-17
Numeric Outputs .....	10-15
Numeric specifier .....	5-18

## o

OFF EXT SIGNAL statement .....	7-22
OFF HIL EXT statement .....	16-4
OFF KBD statement .....	11-27, 11-28
ON CDIAL statement .....	7-1
ON END statement .....	7-1
ON ERROR statement .....	7-1, 14-19
ON EXT SIGNAL statement .....	7-1, 7-22
ON HIL EXT statement .....	7-2, 16-4
ON INTR statement .....	7-2, 12-17
ON KBD statement .....	11-26, 11-27, 11-28
ON KEY statement .....	7-2
ON KNOB statement .....	7-2, 11-25, 11-30
ON TIMEOUT statement .....	7-2, 7-21
ON/OFF CDIAL statement .....	16-26
ON/OFF KBD statement .....	16-23, 16-24
ON/OFF KEY statement .....	16-23
ON/OFF KNOB statement .....	16-25



Operating Parameters, RS-232C .....	13-6
Outbound Control Blocks, Datacomm .....	14-4
Outbound Datacomm Data Messages .....	14-5
Outbound pipes .....	3-21
Outbound transfer .....	9-2
Outbound Transfers, Inbound and .....	9-2
Output .....	2-2
OUTPUT and ENTER and Buffers .....	9-13
Output Area and the Display Line .....	10-5
OUTPUT statement .....	2-18, 2-20, 3-2, 4-2, 5-1
Output to the CRT .....	10-15
OUTPUT USING statement .....	4-10
Output-Area Memory .....	10-23
Outputs that Use Images .....	4-10
Outputting Data .....	4-1
Overrun error (RS-232C) .....	13-4, 13-14

## p

PAIRS conversions .....	8-13
Parallel Poll, Conducting a .....	12-20
Parallel Poll Responses, Configuring .....	12-19
Parallel Poll Responses, Disabling .....	12-20
Parallel Polls, Responding to .....	12-42
Parity bit, RS-232C .....	13-3
Parity Enable (RS-232C) .....	13-7
Parity error (RS-232C) .....	13-4, 13-14
Parity Generation and Checking .....	8-16
Parity option:	
EVEN .....	13-3
NONE .....	13-3
ODD .....	13-3
Parity options, Datacomm .....	14-3
Parity, RS-232C Character Format and .....	13-11
Parity Sense (RS-232C) .....	13-7
PARITY statement .....	8-16
Pass Control message .....	12-22
Passing Control .....	12-34
Path name, I/O .....	3-7
Pen Colors, Changing .....	10-32
Pen Colors in Display Regions, Changing .....	10-8
Peripheral Status line (PSTS) .....	15-43

Pipe .....	3-16
Pipe, named .....	3-17
Pipe, unnamed .....	3-16
Pipeline .....	3-16
Pipes .....	3-15
Plotting Selected Locations on a Touchscreen .....	16-47
Pointers, Buffer .....	9-8
PPOLL CONFIGURE statement .....	12-12, 12-19
PPOLL statement .....	12-12, 12-20
PPOLL UNCONFIGURE statement .....	12-12, 12-20
Premature termination of TRANSFERS .....	9-36
Previews, Chapter .....	1-3
Primary address .....	3-5
Primary addresses .....	12-3
Primary function of an interface .....	2-4
PRINT ALL mode .....	11-8
PRINT PEN statement .....	10-8, 10-32
PRINT position .....	10-25
Priority, Changing System .....	7-8
Priority, Hardware .....	7-10
Priority, Software .....	7-6
Private Telecommunications Links .....	14-15
Processes .....	3-15
Program control (RS-232C) .....	13-9
Program flow (RS-232C) .....	13-12
Protocol .....	14-3

## r

Radix specifier .....	4-14
Re-Assigning I/O Path Names .....	3-8
Re-Directing Data .....	3-11
READ LOCATOR statement .....	16-25
Reading a Screen Line .....	10-28
Reading the Entire Output-Area Memory .....	10-29
Real Numbers, ASCII Representation of .....	2-17
Real Numbers, Internal Representation of .....	2-17
Real Numbers, Representing .....	2-17
Received BREAKs .....	13-4
RECORDS parameter .....	9-16
Records, Transferring .....	9-16
Registers .....	2-19, 6-1
Registers, Buffer-Type .....	9-7

## Registers:

Interface .....	6-2
I/O Path .....	6-5
Relative Positioners .....	16-24
Remote Control of HP-IB Devices .....	12-13
Remote Enable Line (REN), HP-IB .....	12-51
Remote message .....	12-22
REMOTE statement .....	12-12, 12-13
Removing windows .....	17-6
Repeat and Delay Intervals .....	11-9
Repeat Factors .....	5-26
Repeat Factors, Image .....	4-22
Repeatable specifier .....	4-22, 5-26
Representing Real Numbers .....	2-17
RESET statement .....	9-21
Resetting the Datacomm Interface .....	14-11
Resource, specifying a .....	3-2
RESUME INTERACTIVE statement .....	11-31
RETURN attribute .....	8-18
Ring Indicator (RI), RS-232C .....	13-6
<b>rmbxfr</b> .....	9-31
Rotary Control Knob .....	11-30
RS-232C character format .....	13-2
RS-232C Character Format Parameters .....	13-7
RS-232C compatible cables .....	13-34
RS-232C Data Error Detection and Handling, Incoming .....	13-14
RS-232C Data Transfers Between Computer and Peripheral .....	13-5
RS-232C DTE and DCE cable configurations .....	13-18
RS-232C Error Detection .....	13-4
RS-232C framing errors .....	13-4
RS-232C Handshake and Baud Rate .....	13-6
RS-232C, List of Signals .....	14-23
RS-232C Modem Control Register .....	13-16
RS-232C Modem Handshake Lines .....	13-16
RS-232C operating parameters .....	13-6
RS-232C Optional Circuit Driver/Receiver Functions .....	14-21
RS-232C overrun errors .....	13-4
RS-232C parity bit .....	13-3
RS-232C parity errors .....	13-4
RS-232C received BREAKs .....	13-4
RS-232C Serial Interface .....	2-8, 13-1
RS-232C:	



Clear to Send (CTS) .....	13-6
Data Carrier Detect (DCD or CD) .....	13-6
Data Set Ready (DSR) .....	13-6
Ring Indicator (RI) .....	13-6

## S

Screenwidth, determining .....	10-7
Scrolling, Disabling .....	11-8
Scrolling the Display .....	10-26
Secondary Addressing .....	12-11
Select codes (of built-in interfaces) .....	3-4
Select Codes of Optional Interfaces .....	3-4
Selectors, Device .....	3-3
Selectors, HP-IB Device .....	3-5
Semicolon separator .....	4-4
SEND statement .....	12-12, 12-27
Separator, Comma .....	4-4
Separator, semicolon .....	4-4
Serial configuration for BASIC/UX .....	13-7
Serial interface .....	13-1
Serial Interface Programming .....	13-6
Serial Interface, RS-232C .....	2-8
Serial interface (with TRANSFER) .....	9-35
Serial Poll, Conducting a .....	12-20
Serial Polls, Responding to .....	12-44
Series 300 Built-In 98644 Interface .....	13-35
Service request, HP-IB .....	12-41
Service Request Line (SRQ), HP-IB .....	12-52
Service Request message .....	12-22
Service Request (SRQ) .....	12-16
Service Requests .....	7-17
Service routine .....	7-5
Shells .....	3-15
Shift and Control Keys .....	11-3
Sign specifier .....	4-14
Signal functions, RS-232C .....	13-18
Signed Integers, Representing .....	2-15
Simultaneous TRANSFERs .....	9-34
Softkey Label Colors .....	10-36
Softkey Labels .....	10-34
Softkeys .....	11-24



Softkeys and Knob Rotation .....	11-30
Software priority .....	7-3, 7-6
Special-Character Images .....	4-20
Specifiers:	
Binary .....	4-18
Character .....	4-17
Digit .....	4-14
Exponent .....	4-14
Numeric .....	5-18
Radix .....	4-14
Repeatable .....	4-22
Sign .....	4-14
Special-Character .....	4-20
Termination .....	4-21
Specifying an I/O resource .....	3-2
Speed, Execution .....	3-10
SPOLL statement .....	12-12, 12-20
SRQ Interrupts, HP-IB .....	12-17
SRQ Interrupts, Servicing HP-IB .....	12-17
Standard error .....	3-16
Standard input .....	3-15
Standard output .....	3-15
Start bits, Datacomm .....	14-3
Statement-Termination Modifiers .....	5-24
Status Bit message .....	12-22
Status Byte message .....	12-22
STATUS statement .....	6-2
Stepwise refinement .....	8-37
Stop bits, Datacomm .....	14-3
Stop Bits (RS-232C) .....	13-7
String Data, Entering .....	5-8
String Format, Standard .....	4-3
String images .....	4-17, 5-19
String Variables, Entering Data From .....	8-30
String Variables, Outputting Data to .....	8-25
String-variable names .....	3-2
<b>stty</b> .....	13-8
SUSPEND INTERACTIVE statement .....	11-31
Suspended Transfers .....	9-30
Switched (Public) Telephone Links .....	14-15
System controller .....	12-8, 12-32
SYSTEM PRIORITY statement .....	7-8

SYSTEM\$("CRT ID") function .....	10-7
SYSTEM\$("SERIAL NUMBER") .....	16-28

## t

tee command .....	3-19
Telecommunications Links, Private .....	14-15
Telephone Links, Switched (Public) .....	14-15
Terminating a Transfer .....	9-20
Terminating Enters that Use Images .....	5-22
Termination Conditions, Default .....	5-22
Termination Conditions, Multiple .....	9-16
Termination specifier .....	4-21
Terminology .....	2-1
Time gap, Datacomm .....	14-3
Timeout Events, Setting Up .....	7-20
Timeout limitations .....	7-20
Timeout service routines .....	15-27
TIMEOUT time parameter .....	15-26
Timeouts, Datacomm .....	14-12
Timeouts, Interface .....	7-20
Timeouts, Interrupts and .....	7-1
Timing Compatibility .....	2-6
Top-Down Approach, Taking a .....	8-32
Touchscreen, Using a .....	16-46
TRANS binary .....	9-1
Transfer Event-Initiated Branching .....	9-18
Transfer examples .....	9-22
Transfer Formatting .....	9-13
TRANSFER in BASIC/UX .....	9-31
TRANSFER, locking an interface .....	9-31
Transfer methods .....	9-12, 9-32
Transfer parameters .....	9-14
TRANSFER Records and Termination .....	9-17
Transfer restrictions .....	9-3, 9-33
Transfer Sources and Destinations, Supported .....	9-3
TRANSFER statement .....	9-1, 9-9, 9-10, 9-12, 9-35, 15-37
Transfer status .....	9-13
Transfer techniques .....	9-1
Transfer Termination .....	9-13, 9-20
TRANSFER termination, premature .....	9-36
Transferring a Specified Number of Bytes .....	9-15

Transferring Records .....	9-16
Transfers, burst .....	9-12
Transfers, Continuous Non-Overlapped .....	9-15
Transfers, DMA .....	9-12
Transfers, Inbound and Outbound .....	9-2
Transfers Indefinitely, Continuing .....	9-14
Transfers, Non-Overlapped .....	9-15
Transfers, RS-232C Data .....	13-12
Transfers, Suspended .....	9-30
Transfers, The Purpose of .....	9-1
Trapping Function Box key presses .....	16-38
Trapping HP-UX signals from BASIC/UX .....	7-22
Trapping keystrokes .....	11-27
Trigger message .....	12-22
TRIGGER statement .....	12-13, 12-15
Types of Events .....	7-1

## U

Unified I/O .....	8-25
Unnamed buffer .....	9-5
Unnamed Buffers, Assigning I/O Path Names to .....	9-6
Unnamed pipe .....	3-16

## V

Variable names (syntax of) .....	3-6
----------------------------------	-----

## W

WAIT FOR statement .....	9-19
WAIT parameter .....	9-19
wc command .....	3-18
who command .....	3-18
Window stack .....	17-10
Windowing operations .....	17-1
Windows, clearing the contents of .....	17-10
Windows, copying data between .....	17-12
Windows in a window stack, lowering .....	17-10
Windows in a window stack, raising .....	17-10
Windows, listing .....	17-4
Windows, moving .....	17-7
Windows, outputting graphics to .....	17-8
Windows, removing .....	17-6

WORD attribute .....	8-6, 8-7, 8-11
WRITEIO statement .....	7-16









**HEWLETT  
PACKARD**

**HP Part Number  
98796-90030**

Microfiche No. 98796-99030  
Printed in U.S.A. E0988



**98796-90630**

For Internal Use Only